Ejiroghene Precious Oghojafor - Capstone Report

Machine Learning Engineer Nano-degree - September 2019

Classify seedling species from an image

Project Overview

Recently, the state of agriculture and the amount of work people need to put in to check if plants/food is growing correctly is phenomenal, so how do we 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 field of research where this project is derived is Computer Vision. Computer Vision is an interdisciplinary field of science that aims to make computers process, analyse images and videos and extract details in the same way a human mind does. Deep Neural Networks have a (DNN) have a great capabilities for image pattern recognition and is widely used in Computer Vision algorithm, and Convolutional Neural Network (CNN) is a class of DNN which is mostly commonly applied to analysing visual imagery. This methodology can be improved and also used for human detection which can be applied to various disciplines such as self driving cars and robotics. This can also be used in image classification in photos. Here is a gentle introduction to Convolutional. Neural network for image classification. There have also been some research papers that attempt to use using deep learning for image classification.

My Capstone Project classifies 12 different species of plant by image analysis. This project utilizes the dataset from Kaggle Plant Seedling Identification competition. The dataset is also hosted by Kaggle consisting of both train and test data.

The training and test set contains images of plant seedlings at various stages of growth. Each image has a filename that is its unique Id. The dataset comprises 12 plant species. Some of them include, Black-grass, Charklock, Cleavers, Common Chickweed etc. Also each image has a colour layer. In the train dataset I have: 12 species in different directories each containing the following species and number of images

Black-grass: 263, Charklock: 390, Cleavers: 287, Common Chickweed: 611, Common Wheat: 211, Fat hen: 475, Loose Sliky-bent: 654, Maize: 221, Scentless Purse: 231, Small-flowered craneshill: 496, Sugar beet: 385. While the test data consist of 794 unlabelled data. I intend to use about 0.1 split for the train data.

Problem Statement:

The problem being investigated here is a classification of seedling from an image using Convolutional neural Network (CNN). This can help improve crop yield and better stewardship of the environment.

Details of Workflow

1. *Explore the data: This involves importing the dataset, view samples of training data, number of images in the train dataset, species and image count.*
2. *Data Preprocessing: Here is will try to resize, normalise my images*
3. *Model Building: I will try to build a basic CNN and then, I will use transfer learning on top of VGG16 or Inception V3 network. This is because using a pre-trained network is efficient in CNN.*
4. *Algorithm Building: Once I have my model I will try to create an image classification algorithm that load and predict class using my model.*

Metrics

The F1 score was used as my evaluation metrics. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

Since the data was unbalanced using accuracy will be misleading. The F1 score was an optimal alternative that works well when classes are unbalanced. It ensures that images selected to be of a certain class are relevant (Precision) and that relevant images are selected (Recall)

Additionally i needed a loss function to use during training. The loss function should be differentiable and provide direction for optimizing weights. I have chose Categorical Cross Entropy for my training

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

where ŷ is the predicted value.
Categorical cross entropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

A low value for Category cross entropy means that our model has a high probability of predicting target values that align with the actual value.

Analysis:

Data Exploration

Below are five samples species in the train dataset

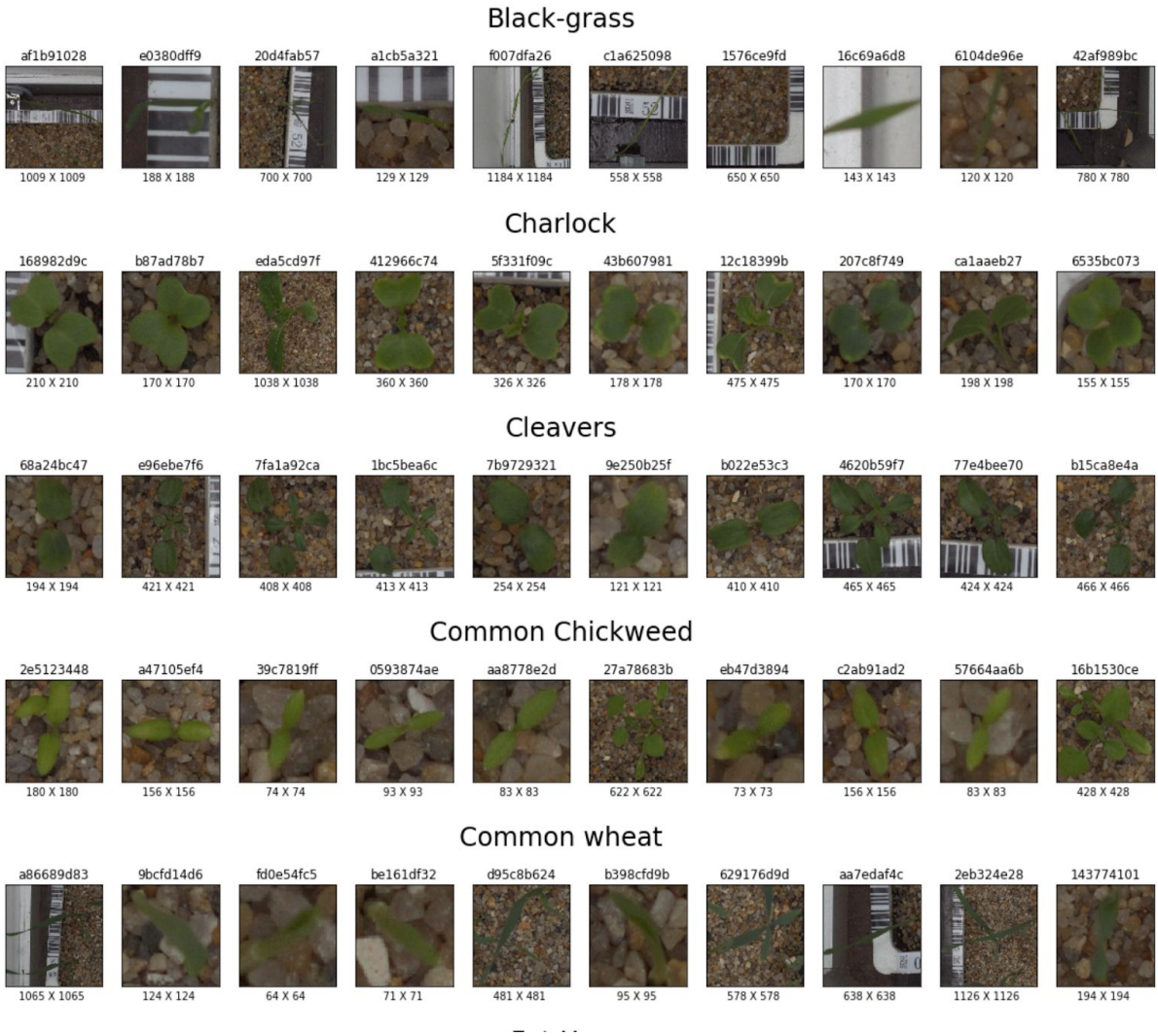

Figure 1: Sample Plant Species class

From a bird's eye view you will notice that Charlock has a “butterfly pattern” while common weed has two light green leaves, But then the species are hard to tell.

You will also notice that for instance Balck grass have a reference scale (looks like barcode). These allow the viewer to understand the actual physical size of the object in the image. It is important to note that my model might see this as a distinguishing feature which might affect the model performance with dataset from the wild.

The images size are not the same across all the dataset some are about 1000 px square will others are less. In order to use a simple Convolutional Neural Network, I had to rescale to a common size. For this i have chosen 224 x 224 px.

After loading the train dataset I examined the class distribution, images below

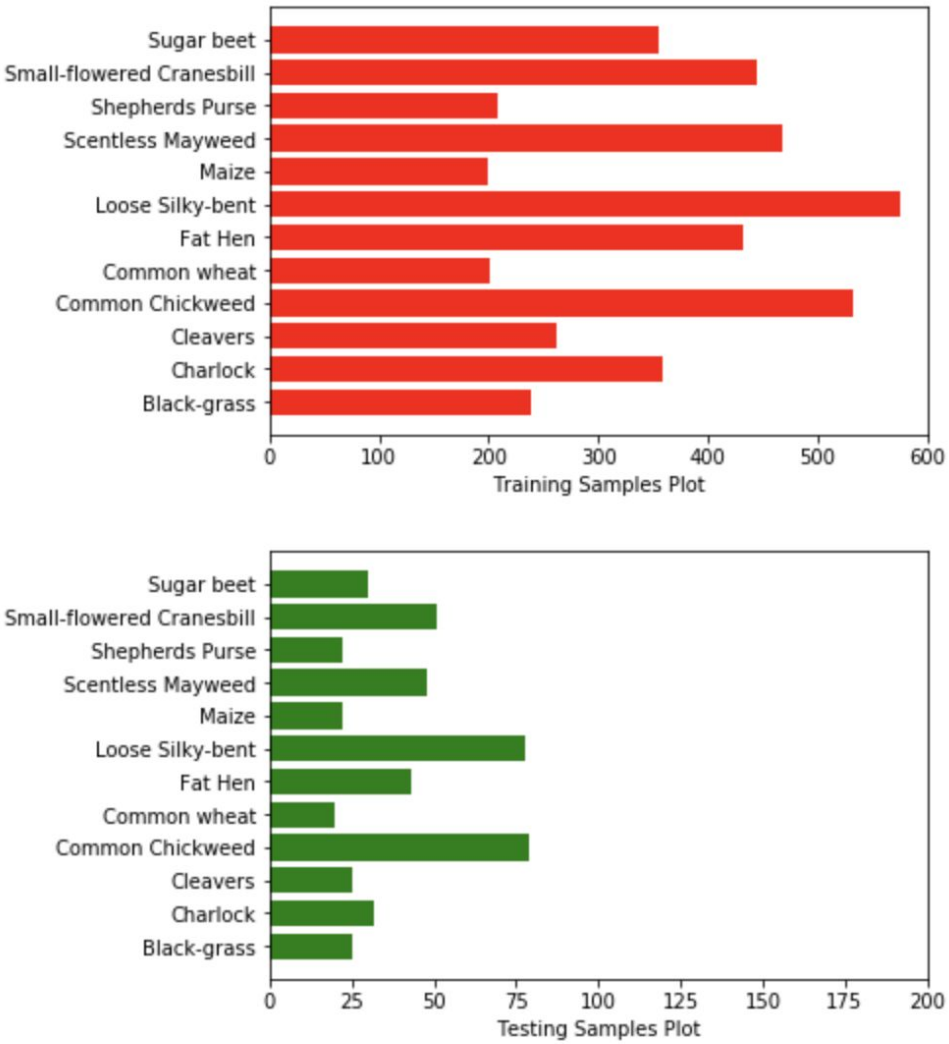

Figure 2: Distribution of data for train and test split

After reading this I decided to do some data augmentation and resampling to account for the imbalanced class distribution

Algorithm and Techniques

In order for me to build a model i tried to do the following

Build a Convolutional Neural Network from scratch

Use transfer learning on a VGG18 network

Use Transfer learning on an InceptionV3 network

For each of theses CNN model there is a series of convolutional layers. They are being initialized with random weights. Each layer serves a purpose which include discovering off latent features in the image. Eealy layers discover abstract features like color and edges. Later layers learn shapes like lines, and the final layer discover actual features in the dataset like stem and leaves.

To build a CNN from scratch I used Keras. Below is a summary of the model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_301 (Conv2D)          (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_302 (Conv2D)          (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_303 (Conv2D)          (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 27, 27, 64)        0
_________________________________________________________________
conv2d_304 (Conv2D)          (None, 26, 26, 128)       32896
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_305 (Conv2D)          (None, 12, 12, 256)       131328
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_306 (Conv2D)          (None, 5, 5, 512)         524800
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 2, 2, 512)         0
_________________________________________________________________
global_average_pooling2d_4 ( (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 12)                6156
=================================================================
Total params: 705,724
Trainable params: 705,724
Non-trainable params: 0
_________________________________________________________________
```

Figure: 3a Model built from scratch

```
model_cnn      = Sequential()
model_cnn.name = 'model'

# Architecture.
model_cnn.add(Conv2D(filters = 16, kernel_size = 2, strides = 1, input_shape=(224,224,3)))
model_cnn.add(MaxPooling2D(pool_size=2))
model_cnn.add(Conv2D(filters = 32, kernel_size = 2, strides = 1))
model_cnn.add(MaxPooling2D(pool_size=2))
model_cnn.add(Conv2D(filters = 64, kernel_size = 2, strides = 1))
model_cnn.add(MaxPooling2D(pool_size=2))

# Added these two layers to the hinted solution
model_cnn.add(Conv2D(filters = 128, kernel_size = 2, strides = 1))
model_cnn.add(MaxPooling2D(pool_size=2))

# Added additional layers to the hinted solution
model_cnn.add(Conv2D(filters = 256, kernel_size = 2, strides = 1))
model_cnn.add(MaxPooling2D(pool_size = 2))

# Added additional layers to the hinted solution
model_cnn.add(Conv2D(filters = 512, kernel_size = 2, strides = 1))
model_cnn.add(MaxPooling2D(pool_size=2))

model_cnn.add(GlobalAveragePooling2D())
model_cnn.add(Dense(12, activation='softmax'))
model_cnn.summary()
```

Figure: 3b Model built from scratch

From the above code snippet we have an input shape of 224 x 224 x 3 tensor representing the three color layers of the resized image. I used 2 x 2 filter throughout the network and a stride set to 1.

For the transfer leaning i used both VGG19 and Inception V3. The VGG19 network is quite similar to the model I built from scratch. It uses 3x3 convolutional window rather than 2x2 window, but it still retains the 2x2 polling layers. It has six more layers than my initial model. For the inception V3 network although the number of layers are a bit difficult, it succeeded in having a smaller footprint in the size of its weights

According to Prakash Jay on Transfer learning he is his recommended approach

- Remove most of the pre-trained layer from the Inception V3/VGG19 network, retaining only the beginning of the network which detects generic aspects of images - like edges and shapes

- Add a new fully connected layer with 12 nodes which implies I should match the number of classes in the seedling dataset
- Randomize the weights of the my new fully connected layer. Freeze all the weights from retained layers of the inception V3/ VGG19 network
- Train the network to update the weights of the new fully connected layer

Since the transfer learning approach the best of them depends on both the size of the new dataset and on how similar the data si to the original data used to train the network. But my case my dataset is not a very large one. Like the one used to from ImageNet. Francois Chollet in his blog post talked about a few options

- Train a small network from scratch (as a baseline)
- Using the bottlenecks feature of pre-trained network
- Fine-tuning the top layers of a pre-trained network

Applying the fine-tuning I preserved all convolution and pooling layers of VGG19 and InceptionV3 and merely adjust which weights were trained with the original model. To achieve a good result i freeze the weights in the first third of the model and retraining those in the remaining two thirds.

Methodology:

Data Preprocessing

First I load the data from the train dataset into memory, there was a total of 4750 training plant images. I then split this data into training and testing sets.

Images are converted into tensors. I resized them into 224 by 224px high with 3 RGB color layers. Each pixel in the tensor is converted to a scalar value in the range [0,1] by dividing it by 225. Afterwards I converted the training and test targets from integers to 12-item vector.

Using Keras ImageDataGenerator is were able to augment and resample the dataset.

Implementation

From my notebook here are the steps

First I define a function to calculate the F1 score

Scratch CNN Model: Compile a simple Convolutional neural network

VGG19 CNN Model: Compile a CNN based on VGG19 using transfer learning

Inception CNN Model: Compile a CNN based on Inception V3 using transfer learning

Define a function to train my dataset based on any of the above model , I already provided the dataset as a parameter top the function, this saves the best model in a directory so I can reference to it in the evaluate function

Define a confusion matrix function

Define a function to evaluate my model

Define the number of epochs to be 20 this is used to train each of my model.

Train each CNN

Evaluate the results

The hyperparameters for each model are

Number of Epochs = 20

Optimizer = optimizers.SGD(lr=0.0001, momentum=0.9)

Which is a Stochastic Gradient Descent

For VGG19 the layers to freeze was 10 while for InceptionV3 it was 100 this was because I thought it might be useful to retain more layers.

Initial Basic CNN Test F1 Score: 0.85263

```
evaluate(model_scratch)
```

```
Test F1 Score: 0.85263
```

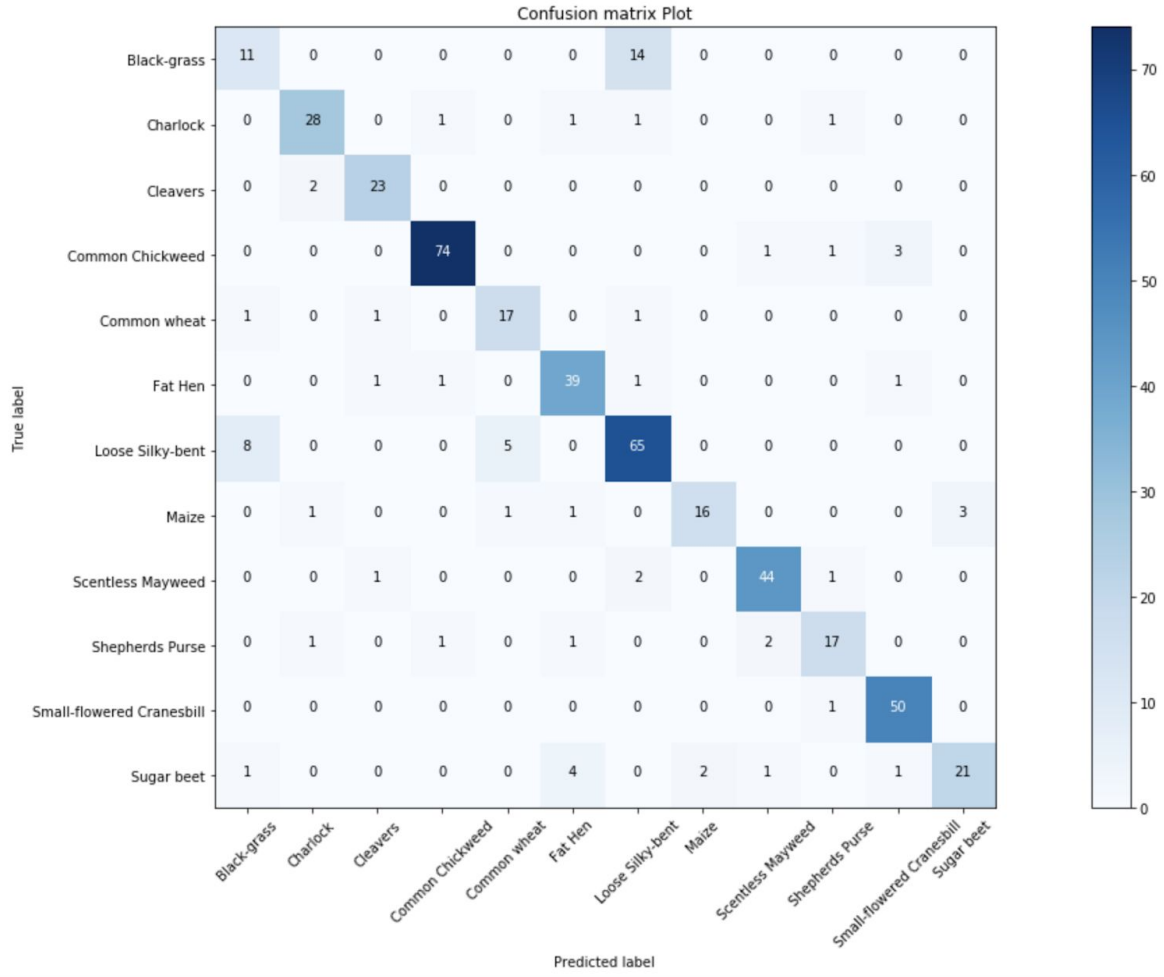

Figure:4 Scratch Model Confusion matrix

VGG19 Model using Transfer learning Test F1 Score: 0.90316

```
evaluate(model_vgg19)
```

```
Test F1 Score: 0.90316
```

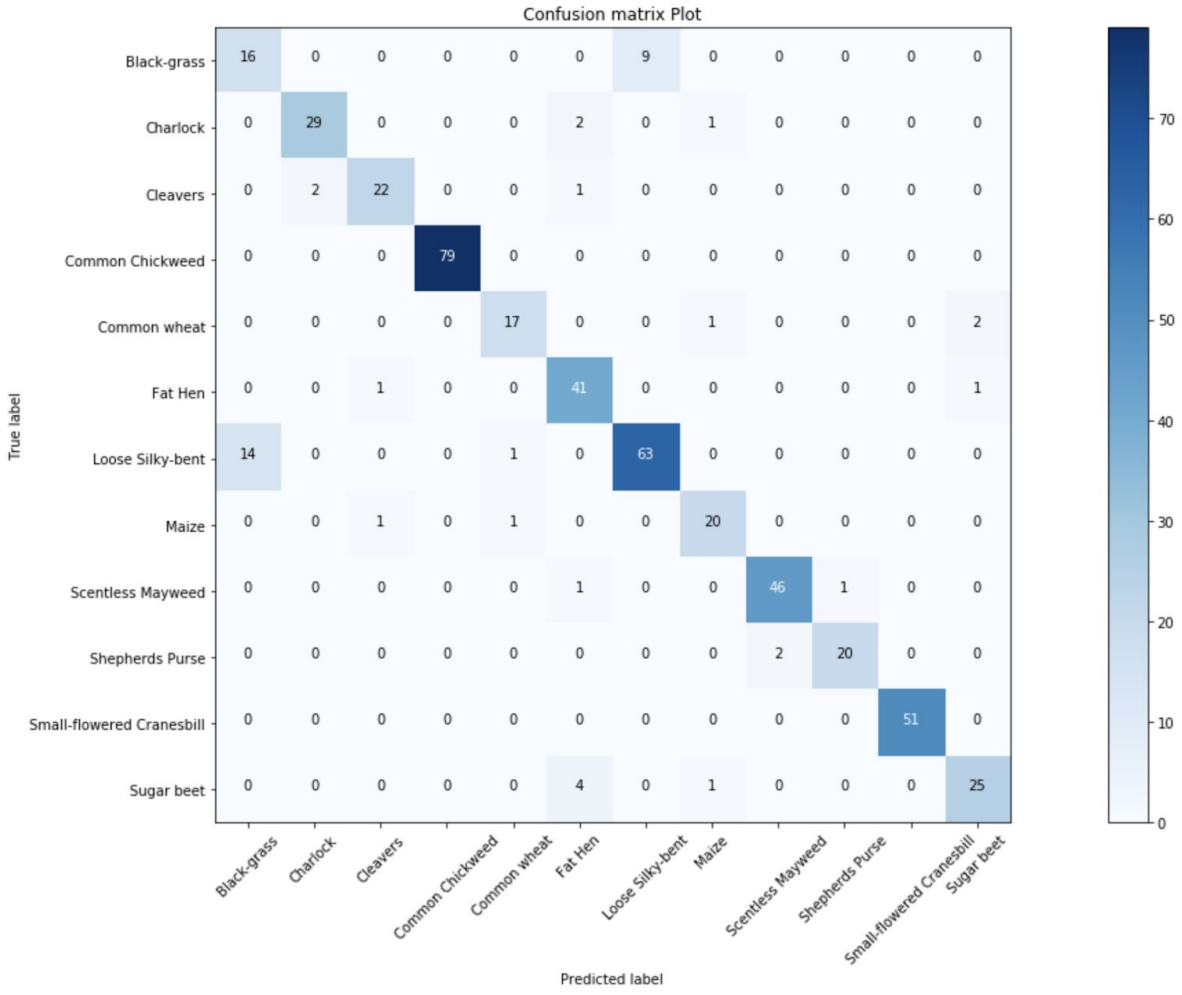

Figure:5 VGG19 Model Confusion matrix

Inception V3 using Transfer Learning F1 Score: 0.92421

```
evaluate(model_inceptionV3)
```

```
Test F1 Score: 0.92421
```

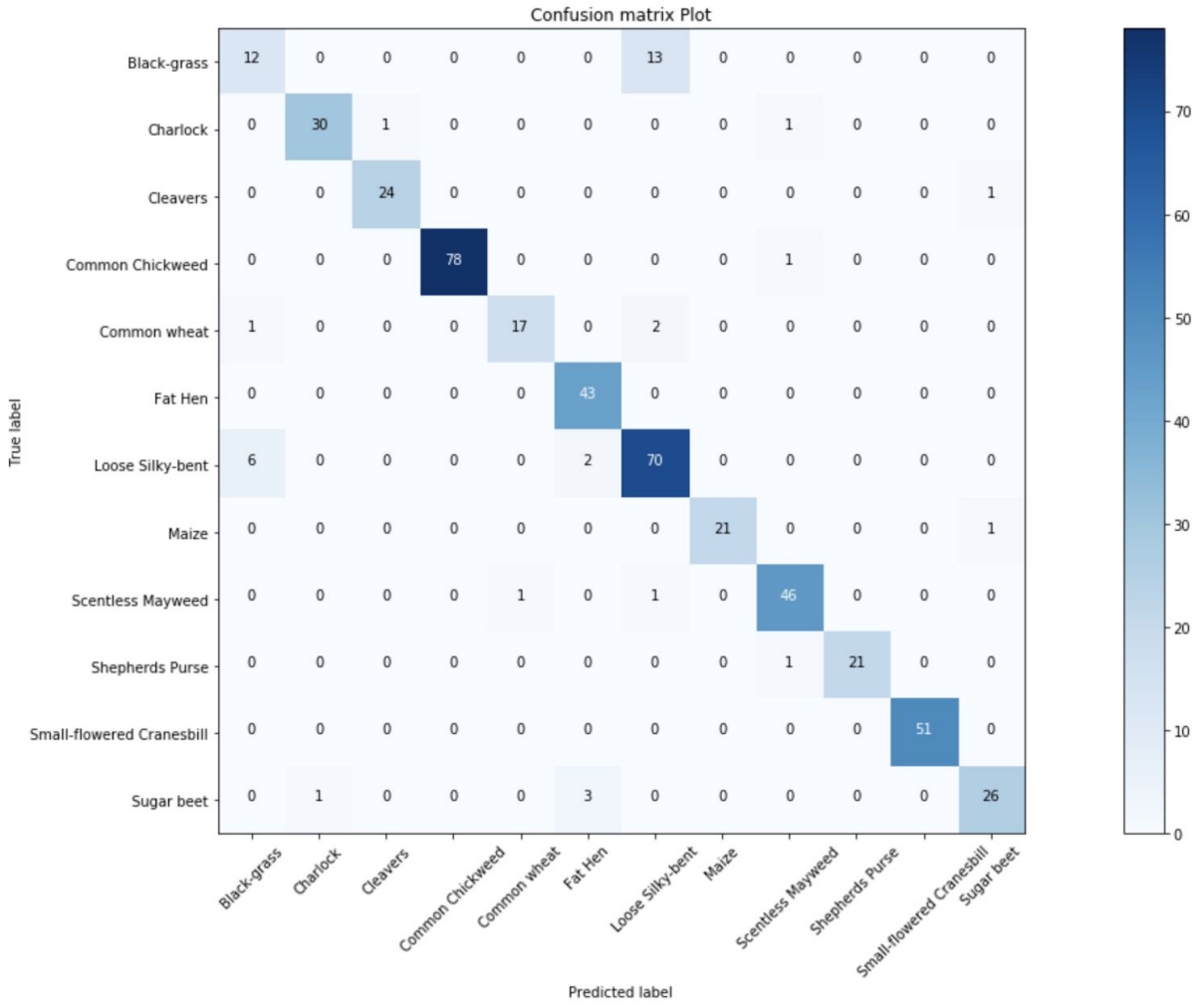

Figure:6 InceptionV3 Model Confusion matrix

You will notice the F1 score for the InceptionV3 model has the highest F1 score 0.92421

As of this the leaderboard in the Kaggle competition has a mean F1 score of 1.0000

For the optimization I used both stochastic gradient descent and rmsprop optimizer (for the model I built from scratch). Rmsprop allows learning rates to differ between parameters and be adjusted dynamically during optimization. Using both optimizer the rmsprop was better for the model in

built from scratch, while for all Transfer learning model it perform more poorly than stochastic gradient descent.

Challenges Overcome:

Since i was using the Udacity dog breed project work space loading the dataset into the environment was a bit challenging ,

Exploring hyperparameters, this seems to be one of the hardest tasks as to building a good model, especially when you have environment is not fast, and this result in the model taking a longer time to train and re-train

Conclusion:

From Figure: 6 which is the confusion matrix for the Inception model with F1 score 0.92421, analysing the p[erforamce of the confusion matrix

You will see that the model did a good job in detecting common chickweed, loose silky bent and small flowered cranesbill.

To improve my model I will likely add additional hyper-parameter tuning, but for the model to perform better in the wild i will need to increase the available dataset, also removing the reference scale from the picture should help the images represent the plant species found in the wild .

References:

1. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy
2. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
4. https://vision.eng.au.dk/plant-seedlings-dataset/
5. https://arxiv.org/abs/1711.05458