Machine Learning Engineer Nanodegree

Capstone Report

Siddharth Baijal
February 16th, 2019

Report

Definition

Project Overview

Weed management is one of the biggest issues in the agricultural domain. It becomes an even bigger issue when a country's GDP heavily depends on agriculture. For many years researchers have tried to find ways to perform site specific weed control. If we are able to identify the plants at an early stage, then it will allow the farmers to perform weedling before the weeds start competing with the crops. Being able to manage weeds over a continuous period of time would result in better harvest in years to come.

This project tries to find a way to effectively differentiate between plant seedlings and weed which can help in weed management.

Problem Statement

The problem is to be able to differentiate weed from a crop seedling. The ability to do so effectively can mean better crop yields and stewardship of the environment [1]. The solution is to be able to identify the plants in one of the 12 categories of plants given in the data set.

Metrics

Evaluation metric used to quantify the performance of the benchmark and the solution model is F1-score. This is the evaluation method as per the Kaggle competition [2] as well. To define F1-score we need to first define the following terms [3]:

True Positive (TP)

A true positive test result is one which detects condition, when the condition is present.

True Negative (TN)

A true negative test result is one which does not detect a condition, when the condition is absent.

False Positive (FP)

A false positive test result is one which detects condition, when the condition is absent.

False Negative (FN)

A false negative test result is one which does not detect a condition, when the condition is present.

Precision

Is the ability of the test to detect a condition when the condition is present.

$$TP / (TP+FN)$$

Recall

Recall is the proportion of positives that correspond to a presence of a condition.

$$TP / (TP+FP)$$

F1-Score

F1-score is the harmonic mean of precision and recall, where F1 reaches the highest value at 1 and the worst at 0.

$$2 * (Precision * Recall / Precision + Recall)$$

Analysis

Data Exploration

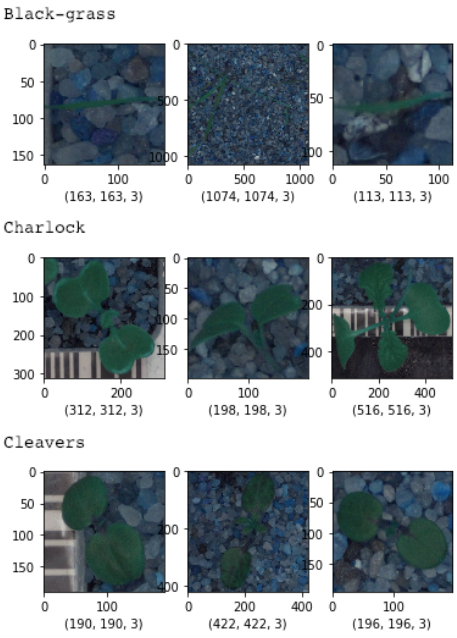
The data set is hosted by Kaggle [1]. The data set consists of training and test set both having images of plants. The images are in png format and are coloured images.

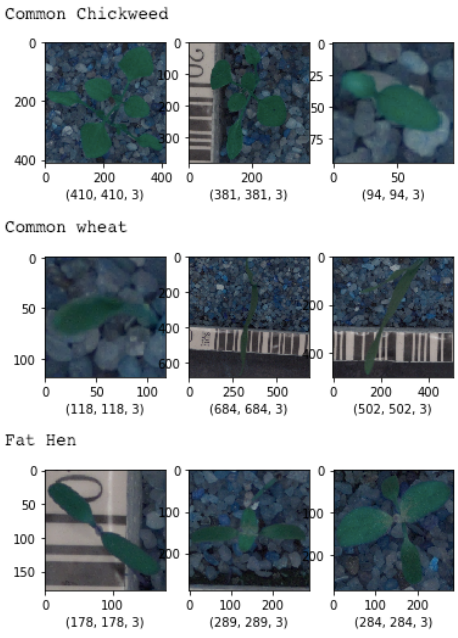
The below description is cited from the Kaggle competition itself.

Data set provided contains a training set and a test set both having 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 competition is to create a classifier capable of determining a plant's species from a photo. The list of species is as follows:

1. Black-grass
2. Charlock
3. Cleavers
4. Common Chickweed
5. Common wheat
6. Fat Hen
7. Loose Silky-bent
8. Maize
9. Scentless Mayweed
10. Shepherds Purse
11. Small-flowered Cranesbill
12. Sugar beet

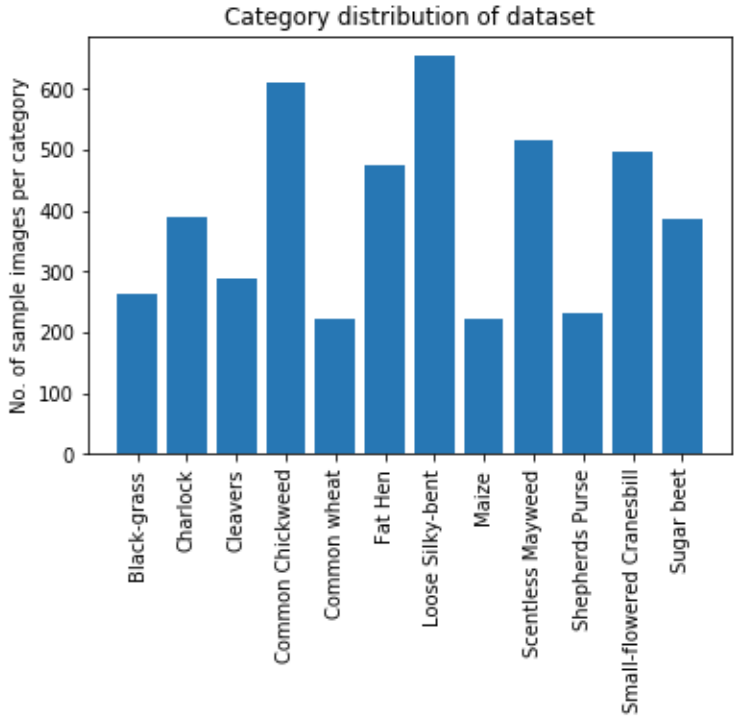
Following are some sample images from the data set:

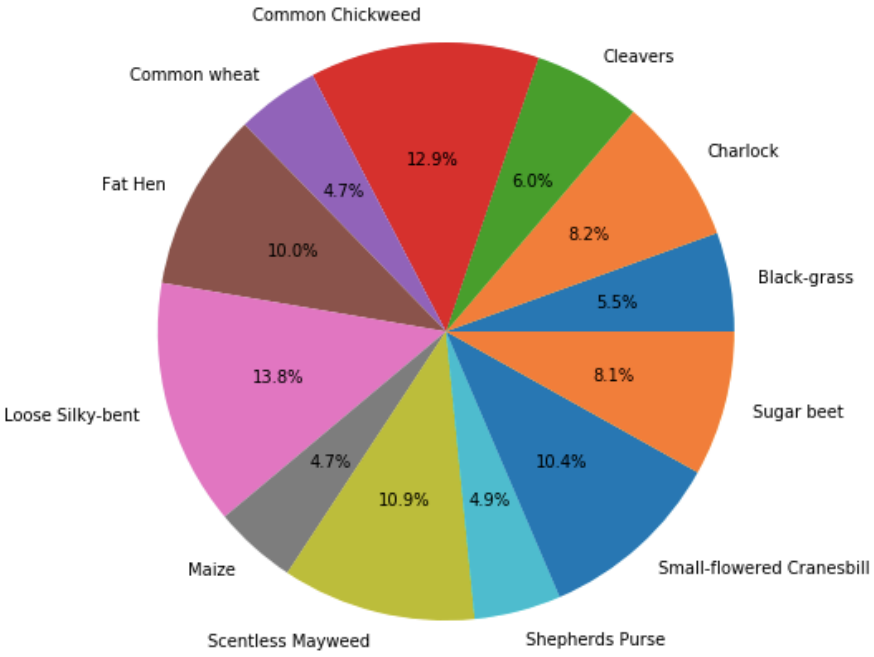




Exploratory Visualization

To find out the data distribution and analyse if the dataset provided is uniformly distributed, I plot the following histogram and the pie chart:





It is clear from the above to plots that the data is not uniformly distributed across different categories and we will have to take it into consideration while designing our solution.

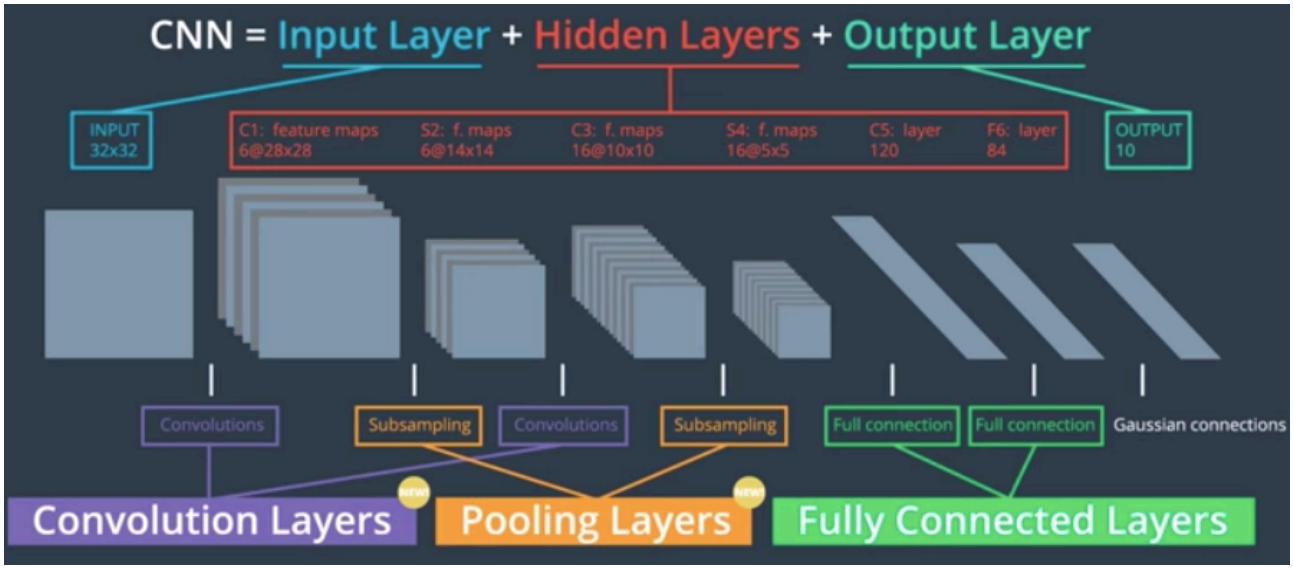
Algorithms and Techniques

I. Convolutional Neural Networks (CNN)

The crux of the solution to this problem lies in designing an image identification mechanism which can identify plant seedlings with high accuracy.
In such cases using a Neural Network model to identify plant images could be great idea. But there are some issues in using Neural Networks. A neural network takes a 1-D vector as input which means we have to convert each image to a 1-D vector. This might lead to losing some of the information from the actual image and thus will not be an effective solution for identifying images.

The above issues can be overcome by using a Convolution Neural Network (CNN). CNN's are built for the purpose of working with or elucidating the pattern in multi-dimensional data. They also take matrix as an input and do not require the image to be converted to a 1-D vector.

A CNN is similar to a Multi layer perceptron (MLP) in a way because the network architecture of the CNN also comprised of stack of layers. They both have an input layer, a set of hidden layer and an output layer. But CNN's differ from MLP's in the types of hidden layers that are included in the model. Following is the description of each layer of a CNN:



Input layer

Input layer is a 3-D layer which is defined by the rows, columns and channels of the image. For example if an image has dimension 224 x 224 and is a RGB image i.e. it has 3 channels, then the input layer is represented as

224 x 224 x 3

Convolutional layer

This is a hidden layer in the CNN. Convolutional layer calculates the dot product of the value of input nodes with their corresponding weight and summing up the result. The calculation is done on a local region and the weights are applied via a filter of certain dimension. The weights in the filter are not constant and are updated through a loss function. If we use 10 filters the output shape of the convolutional layer would be

224 x 224 x 10

Pooling layer

This is the second type of hidden layer in a CNN. The role of the pooling layer is to reduce the dimensionality so that less parameters are required. If we use a max pooling layer with the pool and stride size as 4,4 the the output would be of the following dimension:

56 x 56 x 10

Output layer

The output layer is mostly a fully connected layer having dimension

1 x 1 x N

Where **N** represents the number of categories that are present in the dataset.

II. Image Augmentation

The technique of image augmentation is used to increase the size of the data set. As we saw that our training data is not uniformly distributed across different categories, it is good to use image augmentation so that we can increase the size of the data set os that there is more data to train for our model.

In his technique we perform certain set of actions(like rotations, flips, zoom etc.) on each image while passing it to the CNN model. Thus the model gets to train on more variety of dataset.

III. Transfer Learning

Transfer Learning involves taking a pre-trained neural network and adapting the neural network to a new, different data set. Based on the size and the similarity of the data set from the original data there are 4 main approaches to transfer learning:-

Case 1 : Small dataset and similar data

If the new data set is small and similar to the original data set:

- slice off the end of the neural network
- add a new fully connected layer that matches the number of classes in the new data set
- randomize the weights of the new fully connected layer; freeze all the weights from the pre-trained network
- train the network to update the weights of the new fully connected layer

Case 2 : Small Data Set, Different Data

If the new data set is small and different from the original training data:

- slice off most of the pre-trained layers near the beginning of the network

- add to the remaining pre-trained layers a new fully connected layer that matches the number of classes in the new data set
- randomize the weights of the new fully connected layer; freeze all the weights from the pre-trained network
- train the network to update the weights of the new fully connected layer

Case 3: Large Data Set, Similar Data

If the new data set is large and similar to the original training data:

- remove the last fully connected layer and replace with a layer matching the number of classes in the new data set
- randomly initialize the weights in the new fully connected layer
- initialize the rest of the weights using the pre-trained weights
- re-train the entire neural network

Case 4: Large Data Set, Different Data

If the new data set is large and different from the original training data:

- remove the last fully connected layer and replace with a layer matching the number of classes in the new data set
- retrain the network from scratch with randomly initialized weights
- alternatively, you could just use the same strategy as the "large and similar" data case

Benchmark

For the solution of the problem, I planned on using the Convolutional Neural Network (CNN) with transfer learning to identify plant seedlings. Therefore it made sense to use a CNN created from scratch to use as the benchmark model for this project. This would allow us to see how much greater impact does models created via transfer learning have over models created from scratch.

Following is the summary of the benchmark CNN model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 224, 224, 10)      760
_________________________________________________________________
activation_1 (Activation)    (None, 224, 224, 10)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 10)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 20)        5020
_________________________________________________________________
activation_2 (Activation)    (None, 56, 56, 20)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 30)        15030
_________________________________________________________________
activation_3 (Activation)    (None, 14, 14, 30)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 30)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 270)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2048)              555008
_________________________________________________________________
activation_4 (Activation)    (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                24588
_________________________________________________________________
activation_5 (Activation)    (None, 12)                0
=================================================================
Total params: 600,406.0
Trainable params: 600,406.0
Non-trainable params: 0.0
_________________________________________________________________
```

The benchmark model resulted in an F1-score of **0.7810**.
This was the target value that had to be beaten by the models created via transfer learning.

Methodology

Data Preprocessing

Following set of preprocessing steps have been performed on the dataset:

- ONE-HOT ENCODING
  The categories of plant training data set are one hot encoded using the np_utils.to_categorical method and stored in train_targets variable.

- TRAIN, VALIDATION & TEST SPLIT
  Here we have divided the training dataset into training, validation and test set and finally have the following number of images:

DATA SET	NO. OF IMAGES
Training (72%)	3420
Validation (18%)	855
Test (10%)	475
Total	4750

- RESIZE IMAGES
  In this step we resize each images from each dataset to shape 224 x 224. We also create a 4D tensor for the images which is required by Keras to process images. Each image has the following 4-D tensor of shape

  (1 x 224 x 224 x 3)

- DEFINE IMAGE AUGMENTOR
  Here we define the steps taken in the image augmentation. Although augmentation is not a preprocessing step, I have defined it there for clear understanding what changes will take place in the image when feeding it to the CNN model.
  The following code snippet shows different augmentation steps[4]:

```
# create the data augmenter
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.2,
                         zoom_range=0.2, horizontal_flip=True, vertical_flip=True, fill_mode="nearest")
```

Implementation

The implementation involved first designing the benchmark CNN model. Then using transfer learning build Inception_V3 and Xception models.

Model evaluation has been done on the basis of F1-score. Since there is no default F1-score available in Keras 2.0, I have defined a custom F1-score metric to evaluate the CNN models.

Benchmark Model

For creating the benchmark model, I took the following steps:

- Benchmark model takes input shape as (224,224,3)
- It has 3 Convolution layers. Each convolutional layer is followed by a 'ReLu' activation function and a Max Pooling layer to reduce the dimensionality.
- The result of final max pooling layer is flattened to reduce the dimensionality further.
- At last we use 2 dense layers where the final dense layer has the shape equal to the number of categories of plant seedlings i.e. 12.
- At last there is a softmax activation function to help classify from the 12 categories.
- Using this model resulted in 600,406 trainable parameters.
- Optimizer for benchmark model is **rmsprop** and loss function is **categorical_crossentropy.**

Models created via Transfer Learning

Two models were created via transfer learning namely Inception_V3 and Exception model. As described in the Algorithm and Techniques section, there are 4 cases for Transfer learning. Since plant seedling dataset is relatively small and different from the dataset on which Inception_V3 and Exception models are trained, I followed Case 2: Small dataset, different data.

- For **Inception_V3** model I froze the first 150 layers of the model.

- For **Xception** model I froze the first 75 layers of the model.

- At the end of both the models I added a Global Average pooling layer to reduce the dimensionality and a dense layer of the size of the number of categories

- Optimizer for both the models is **SGD** and loss function is **categorical_crossentropy**.

For all the three (benchmark, incpetion_v3, exception) models, I used Image augmentation defined above to increase the size of the data set on which the models trained. I also saved the weights during the training phase of each model using ModelCheckpoint method and used them later to evaluate the model on the test set. F1-score is used as metrics to evaluate the models.

Complications

Some of the complications that occurred during the implementations were:

- While defining the benchmark model I faced issues as I wanted to keep the benchmark model simple yet effective so that I set a good bar for the transfer learning models. I had to try a lot of different combinations of layers before arriving at the final model.

- When working with transfer learning model, there was no clear cut way regarding how many layers should I freeze. This was a bit of hit and trial for me and I read different blogs to decide on the number.

- As I was using the AWS credits provided by Udacity, I had to be quite cautious around on how I used the GPU resources, this sometime lead to me running code on my local machine till the time I was sure it is okay to run on the GPU enabled AWS machine. But this also helped in understanding why the GPU resources are important and how to effectively utilise them.

Refinement

One of the major refinement steps was using image augmentation. When I created the benchmark model, I was getting a F1-score on the test set of around 0.60-0.65. But I wanted the F1-score of the benchmark to be higher than this. Thus I used image augmentation, so that the benchmark (and other models) got more data to train on. This increased the F1-score a lot and finally yielding it to be around **0.78** for the benchmark model.

Another major refinement step was deciding the number of layers to freeze. Both the transfer learning models were not yielding good results because I had frozen less number of layers in the beginning of my work. For example I froze only 60 layers in inception_v3 and around 40 layers in exception model. The resulting F1-score was not impressive. Finally, I was able to find the correct number of layers to be froze that gave me the desired result. But I still feel that I need to get a better understanding of this rather than hit and trial.

Another refinement step was to use SGD optimizer instead of rmsprop in the transfer learning models. This helped in increasing the F1-score to their current value in the transfer learning models. Using this optimizer was suggested in this blogpost - https://flyyufelix.github.io/2016/10/08/fine-tuning-in-keras-part2.html

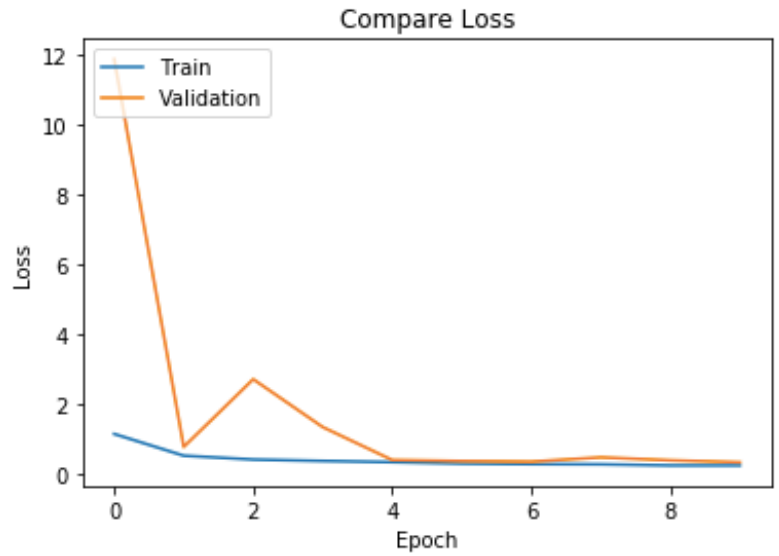
Results

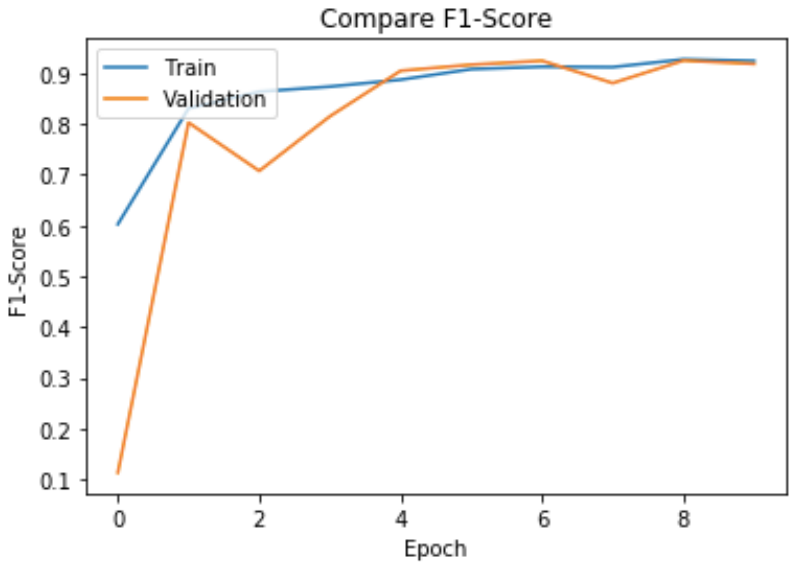
Model Evaluation and Validation

Both the transfer learning models did exceedingly better than the benchmark model. The Xception model achieved F1-score of **0.9010** , but the Inception_V3 model did a little better and achieved F1-score of **0.9157** on the test dataset.

Also, Inception_V3 model recorded a lesser training time than the Xception model. The Inception model completed training in 754 seconds where Xception model completed training in around 981 seconds.

Things get interesting when we plot the comparison of training and validation dataset for both loss and F1-score for Inception_V3 model:





As seen from the above graphs the loss is ~0.22 for both training and validation. But we can see that there is a sudden same in the loss for validation in the second epoch.

Similarly for F1 score ~0.91 for both but again the F1-score drops to ~0.7 in the second epoch and goes higher than the training score around for 4-6 epoch.

Justification

Below is the comparison of three models created to solve the problem.

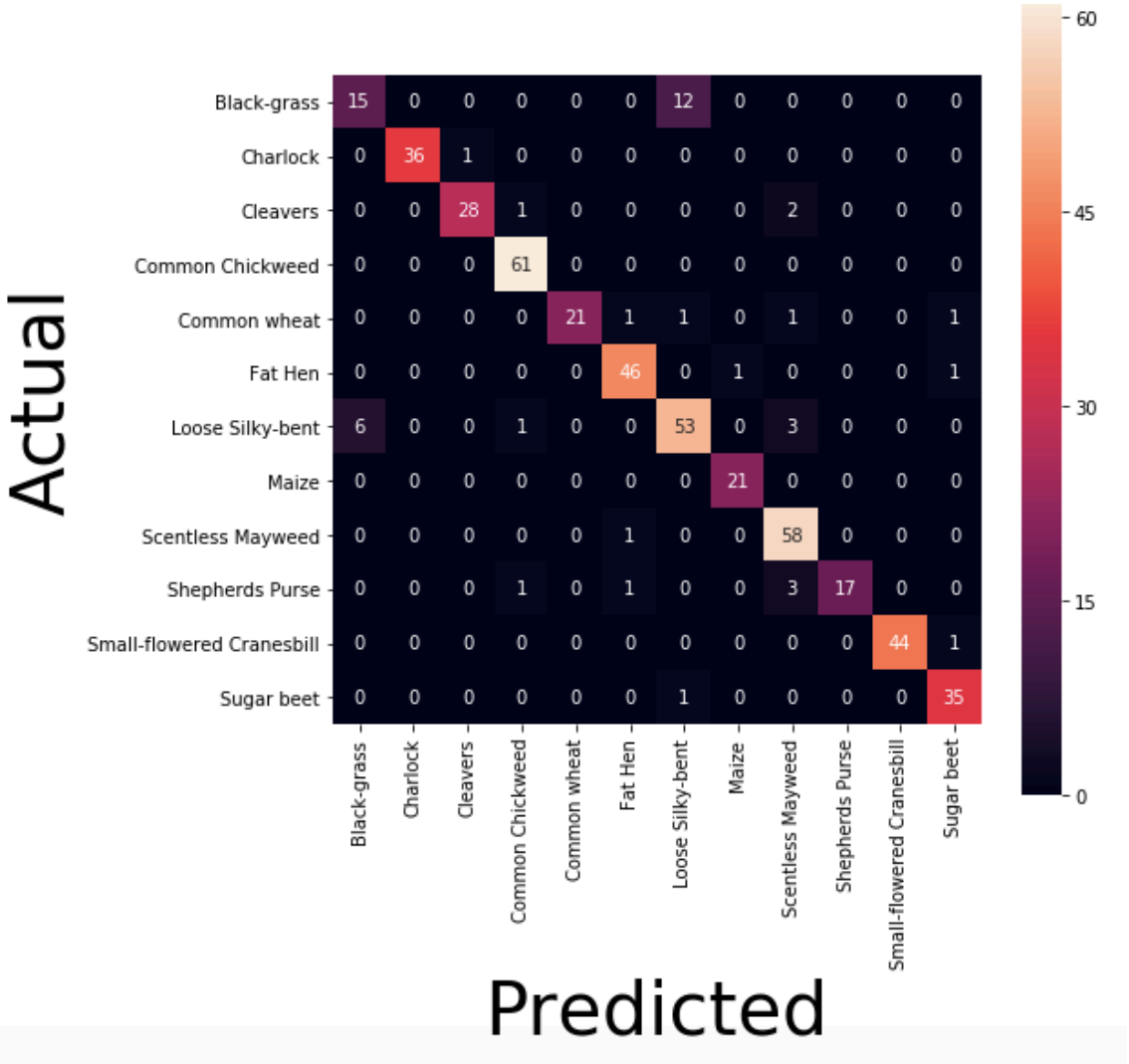
Model Name	F1 Score	Training Time (seconds)
Benchmark	0.7810	262
Xception	0.9010	981
Inception_V3	0.9157	754

As seen from the above image both the transfer learning models beat the score of the benchmark model by a great margin. The **Inception_V3** model is a clear winner with an F1-score of **0.9157.**

Conclusion

Free Form Visualization

To visualize the current state of the Inception_V3 model and to identify segments for further improvement, I created the below confusion matrix:



Most of the plant seedling categories are identified correctly with minor errors, but there seems to be a gap when it comes to **Black grass** and **Loose Silky-bent** categories. 12 Black-grass images have been identified as Loose Silk-bent. Also, 6 Loose Silky-bent images are identified as Black grass.

If I can improve my model on these particular set of categories, then the accuracy of prediction would be much higher.

Reflection

To summarize the solution, I can say that I set out to solve the problem of weed management, which is a big concern in agriculture domain. I have been able to successfully implement the solution to the problem by designing CNN models through transfer learning and have concepts like image augmentation as well.

To solve the problem, I first created a benchmark CNN model from scratch and then worked on improving it to get a good score. Next I created 2 transfer learning models, Inception_v3 and Xception model. While creating these models, it was difficult to identify the number of layers to freeze and required a little hit and trial method. While training the models I saved the weights to use them later. I used these weights to compare between training and validation F1-score and loss. I also used the weights to create a confusion matrix for each model to understand if there is a particular section of categories that are not being identified by the model properly. Finally, I reached a good score with my transfer learning models.

As discussed in the Implementations sections, I faced some complications during the implementation which ranged from tweaking the models to effective resource management when it comes to running code on GPU's. These complications gave me an exposure to how real world problems are being solved and gave me a better understanding of how to work with image classification problems in the future.

Improvement

As seen in Free-Form Visualization section, there is a segment of categories that is not being identified correctly by the Inception_v3 model. This problem needs to be rectified to provide higher accuracy. I think if I made some changes in my augmentation section then the model would be able to train better on those categories and would be able to distinguish between them properly.

As a general improvement to this project, I plan on masking the background noise of the images. Masking the noise would help in giving a clearer input to the models and would result in better training and hence higher accuracy in identifying between the 12 categories.

References

[1] https://www.kaggle.com/c/plant-seedlings-classification/data

[2] https://www.kaggle.com/c/plant-seedlings-classification#evaluation

[3] https://groups.bme.gatech.edu/groups/biml/resources/useful_documents/Test_Statistics.pdf

[4] https://keras.io/preprocessing/image/