Machine Learning Engineer Nanodegree

Capstone Project

Muthudivya Navaneetha Kannan
April 28th, 2018

I. Definition

Project Overview

One of major challenges in agriculture is weed management. Sustainable crop production is the need of the hour as we are struggling to supply food for the increasing global population. For effective crop production, weed management is one of the key factors. The ability to automatically classify and remove weeds among the crop at the seedling level will yield huge benefit and higher harvest. This problem is not something that can be totally eradicated, but a solution to this problem can help solve and mitigate weed problem now and, in the years, to come.

This project is an attempt to tackle one such problem. The dataset used for this problem has been donated by The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, in the hope "*to provide researchers a foundation for training weed recognition algorithms*" [3]. A benchmark for standardizing the results of this classification problem has been discussed in arXiv:1711.05458 using the original dataset.

Problem Statement

Problem is the similarity among different species of plant seedling. In this project, the model should be able to determine the species of a seedling from an image. The dataset contains images of twelve categories of plant seedling, and hence the solution should be able to classify an image into one of twelve categories.

The solution of this problem is to build and train a model that can classify the new unseen image into one of the twelve mentioned categories accurately.

This can be achieved by:

- For any given image, the model outputs one predicted value (i.e., the species of the seedling with highest probability)
- However, the model outputs probability of every class that the image can belong to, we can design the output to be in such a way that the only category with highest probability is shown as prediction for the input image.

Metrics

Since the dataset is unbalanced, the appropriate metric to evaluate the performance of this classifier is by calculating the Mean F Score.

The F1 score [1] 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 relate contribution of precision and recall to the F1 score are equal. The formula for the F1 score is given as:

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

Precision and recall can be calculated as [2]:

Precision = number of TP / (number of TP + number of FP)

Recall = number of TP / (number of TP + number of FN)

Where, TP is True Positives, FP is False Positives and FN is False Negatives.

As this problem statement has multi-class solution, Mean F1 score is the weighted average of the F1 scores of each class.

II. Analysis

Data Exploration

Datasets are obtained from Kaggle [4]. A training set and a test set containing them images of plant seedlings at various stages of grown. Each image has a filename that is its unique ID. The dataset comprises of 12 plant species, which are listed below:

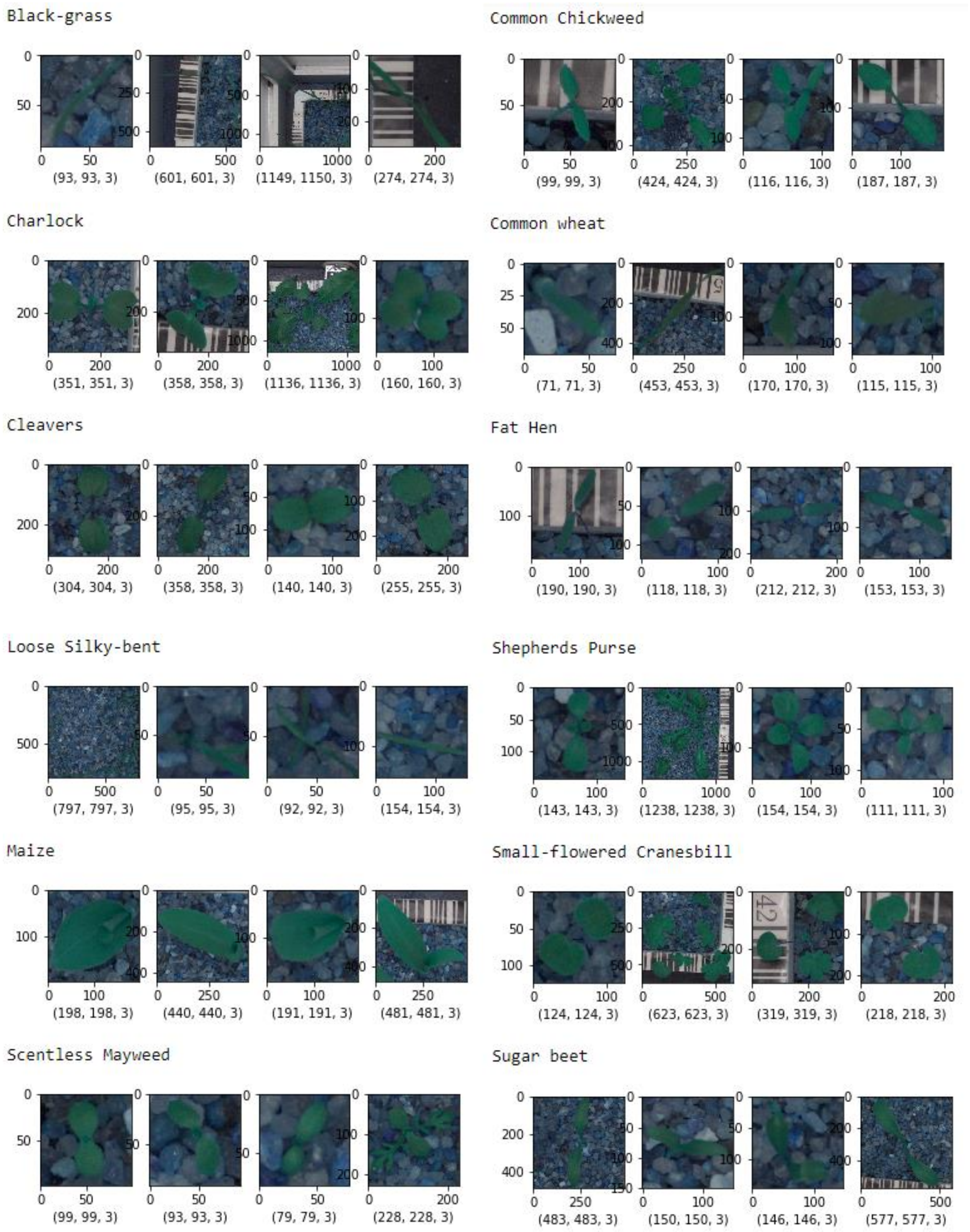
Black-grass
Charlock
Cleavers
Common Chickweed
Common wheat

Fat Hen
Loose Silky-bent
Maize
Scentless Mayweed
Shepherds Purse
Small-flowered Cranesbill
Sugar beet

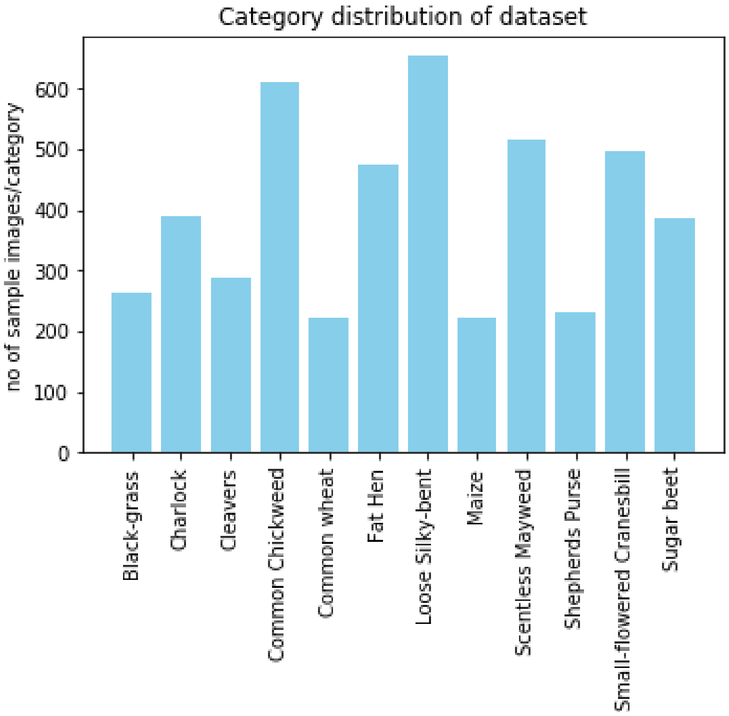
The objective is to build an image classifier that can accurately identify the image and classify the image into one of the above 12 species.
A quick look at the train dataset tells that the number of images available for each class (['Sugar beet', 'Small-flowered Cranesbill', 'Maize', 'Common wheat', 'Common Chickweed', 'Fat Hen', 'Black-grass', 'Cleavers', 'Scentless Mayweed', 'Charlock', 'Loose Silky-bent', 'Shepherds Purse']) is varied (496, 221, 221, 611, 475, 263, 287, 516, 390, 654, 231), i.e. the dataset is highly unbalanced. To combat with the unbalanced dataset, one of the many strategies can be applied. The dataset can be under-sampled, or data augmentation can be applied to balance the under-represented classes. However, the problem of unbalanced dataset can also be dealt by calculating the confusion matrix and F1 score for the classifier, as these metrics give us the more appropriate evaluation of the model. The images are RBG colored and have a background that is undesirable for the problem at hand can be gotten rid of by masking – this can be achieved by converting the RGB images to HSV mode by tuning the related parameters and is discussed in the future work. I intend to use 80:20 as the train to validation split ratio for training my classifier.

Exploratory Visualization

Below are 4 random samples from each of the categories:



Further plotting the number of samples available in the training set for each category gives the below graph:



Above graphical representation of the categorical distribution tells us that the training dataset is unbalanced. Few of the classes have only a little over 200 samples whereas few others have over 500 samples. This shows that dataset is highly unbalanced and hence performance of this classifier can not be measure by its accuracy and hence F1 score is used to measure the performance.

Algorithms and Techniques

In general, Neural networks accept a single vector as input, transform it to a series of hidden layers, which in turn is made up of set of neurons that are fully connected to all neurons in the previous layer. Neurons of the same layer are independent and do not share any connections. After the hidden layers, is the last fully connected layer which is also called the 'output layer', where each node outputs score for each class. The downside of regular neural network is that they don't scale well to full images. Its mainly because with images of decent size, the number of neurons and weights that the network must accommodate becomes unmanageable. This is where Convolutional Neural Network comes to rescue with its neurons arranged in 3 dimensions (width, height, depth).

Each of the layer in CNN accepts 3D input volume and transforms it into 3D output volume. Following is a simple visualization of how CNN arranges its neurons in 3 dimensions (width, height, depth):

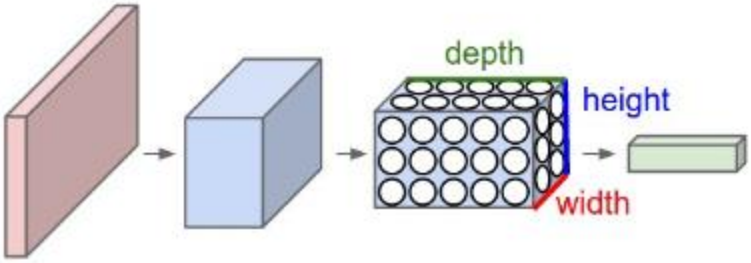


Fig: http://cs231n.github.io/convolutional-networks/

Following are the layers that are used to build a CNN:

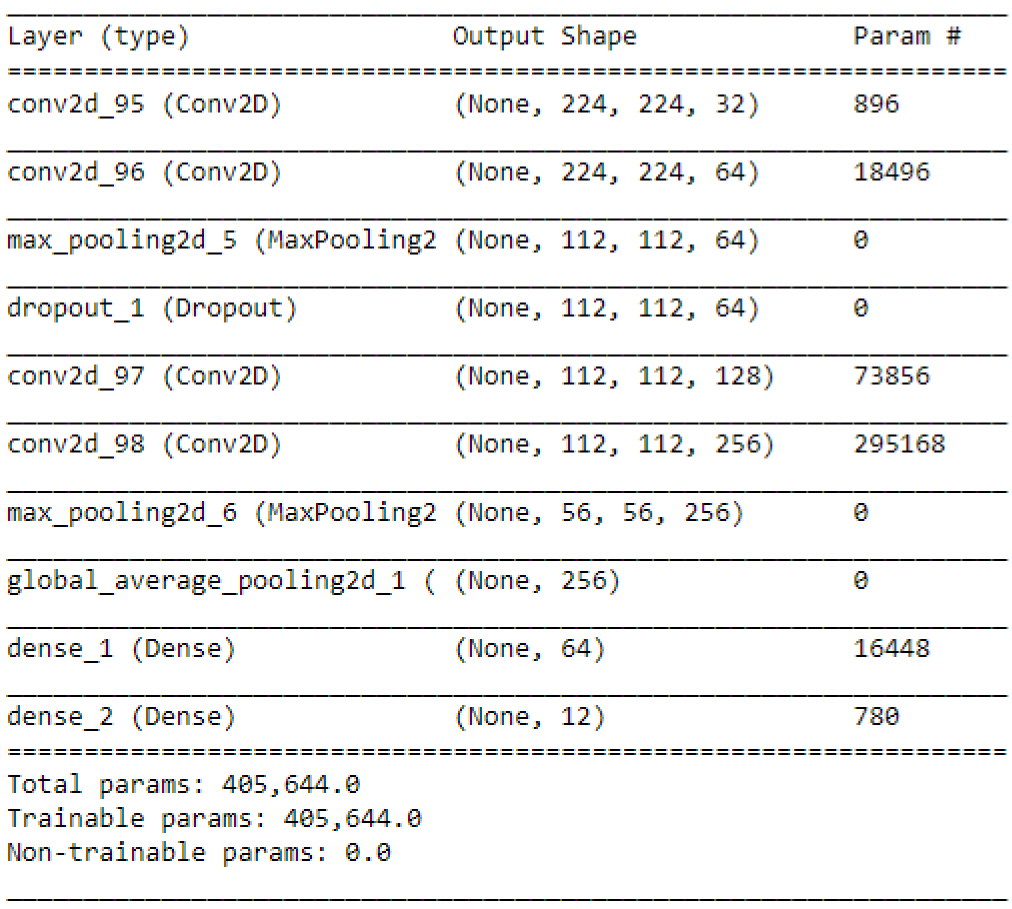
Input layer (w,h,d)	Input layer of shape (w,h,d) represents image of size 'w x h' and 'd' number of color channels. For example, for an image of size (224x224) with 3 color channels (RGB), the input layer will hold raw pixel values of the image as a vector of size [224x224x3]
CONV layer	The CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. For eg, if we use 32 filters, CONV layer will output a volume equal to (224x224x32)
Activation functions	Activation layer will apply an element wise activation functions, leaving the volume unchanged.
POOL layer	Pool layer performs downsampling operation along the spatial dimensions (width, height), outputting a reduced volume than the previous layer. For eg, (112x112x32)
Fully-Connected Layer	FC layer, also called as the dense layer, each neuron will be connected to all the neurons of the previous layer. FC layer when used as the output layer results in much reduced volume of size [1x1xm], where m is the number of categories that are to be predicted. Each of the nodes of fully connected layer outputs a score corresponding to a class score.
Dropout layer	Dropout layer is used as a method of regularization to combat over-fitting of the training set. It 'drops' neurons at random (depending on the probability mentioned) while calculating the forward prop and backward prop, resulting in a simpler version of the CNN for each iteration and hence giving the model a hard time to overfit the training set.

Hence for this problem, I built CNNs keeping in mind that it is an image classification problem.

Benchmark

As my benchmark model, I built a simple CNN architecture, trained it on the training set and measured its performance by calculating the F1 score. With over 405k+ trainable parameters, the model's F1 score is 0.687237026648.

The model has the following architecture:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_95 (Conv2D)           (None, 224, 224, 32)      896       
_________________________________________________________________
conv2d_96 (Conv2D)           (None, 224, 224, 64)      18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 112, 112, 64)      0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0         
_________________________________________________________________
conv2d_97 (Conv2D)           (None, 112, 112, 128)     73856     
_________________________________________________________________
conv2d_98 (Conv2D)           (None, 112, 112, 256)     295168    
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 56, 56, 256)       0         
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448     
_________________________________________________________________
dense_2 (Dense)              (None, 12)                780       
=================================================================
Total params: 405,644.0
Trainable params: 405,644.0
Non-trainable params: 0.0

_________________________________________________________________
```

III. Methodology

Data Preprocessing

Loading the dataset

The dataset was downloaded from Kaggle. The path of the dataset 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' values are one-hot encoded using np.util.to_categorical function and returned as an array of one-hot encoded 'y_targets' vector.

Train-validation-test split

The dataset was first split in the ratio of 85:15 as training-validation and test set. The training-validation set containing 4037 images were further split into training and validation set in the ratio 80:20 giving 3229 images for training set and 808 for validation set. The test set with 713 images and labels pair was kept untouched for final evaluation the classifier.

Implementation

For the CNN that I built from scratch, following are the key properties:

- Tensors of shape (224,224,3) representing the image shape and 3 channels were fed into the network.
- The network comprised of 4 convolutional layers with max pooling layers after every 2 convolutional layer, a global average pooling layer to cut down the dimensionality, and two fully connected layers.
- 'Relu' activation function was applied on the convolutional layers, and sigmoid function was applied on the last fully connected layer with 12 nodes to classify the 12 categories.
- Filters were set in increasing order (32, 64, 128, etc) to achieve more complex pattern as the layers were added, keeping the kernel size =3x3 and with stride of size 1x1
- With this architecture, there were 405,644 trainable parameters
- The model was compiled with custom metric (f1 score) and set to calculate categorical cross-entropy loss for calculating the gradient descent during back propagation.

For CNNs built using Transfer Learning:

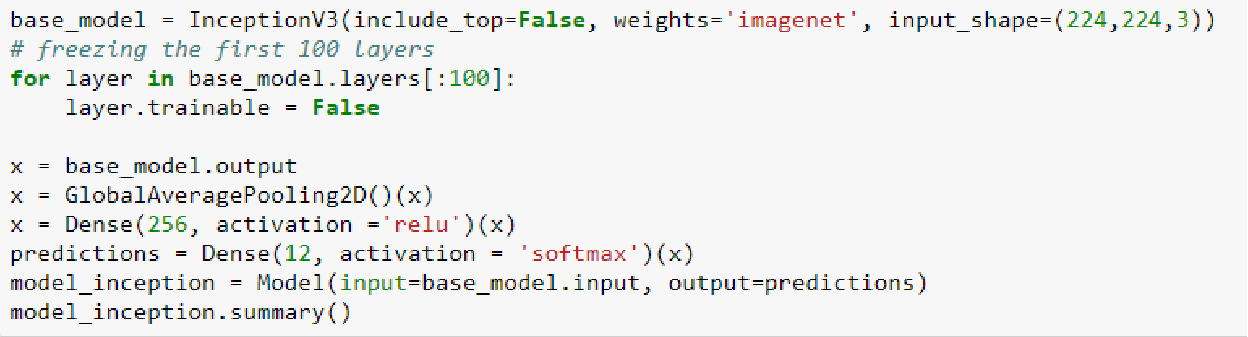
Since the dataset in hand was fairly small than the datasets that were used to train the Inception and Xception networks, and also since the dataset differed from that of pre-trained network, I chose to follow the “Case 2: Small Data Set, Different Data” technique given in the course material.

When using the InceptionV3 as my pre-trained network, I froze the first 100 layers of the pre-trained network, added global average pooling layer, fully connected layers, retained the weights from the pre-trained network and finally, retrained the model to get updated weights for the new network. While using the Xception as my pre-trained network, I removed the last 50 layers, froze the first 75 layers and added few fully connected layers with ‘tanh’ activation function. I noticed during the training that the network was trying to overfit the training data, so I also added few ‘Dropout’ layers in between to avoid over-fitting. Since I had removed a good chunk of layers from the pre-trained network, the training parameters were only about 1,345,628 out of 8,935,540.

As the performance metrics for our problem ‘F1 score’ is not readily available in keras, custom metric function was built to calculate the F1 score, and it was passed as a parameter during the model compilation. Weights from the model were saved using the ModelCheckpoint function, with save_best_only set to True, only the best weights were saved in the .hdf5 format. More on the performance of the model is discussed in section ‘IV. Results’

Refinement

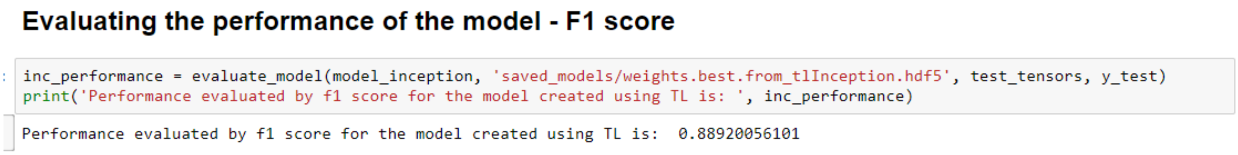
I started this problem with building a CNN model using the InceptionV3 pretrained network where I froze only first 50 -60 layers and added one fully connected layer at the end of the network with 12 nodes to classify the categories. With this architecture the model’s loss could only reach 2.13. In an attempt to bring this further down, I increased the number to layers to freeze to 100, and also added a fully connected layer with 256 nodes just before the final layer with ‘relu’ activation function. Figure below gives the final architecture after refinement for this model:



```
base_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(224,224,3))
# freezing the first 100 layers
for layer in base_model.layers[:100]:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation ='relu')(x)
predictions = Dense(12, activation = 'softmax')(x)
model_inception = Model(input=base_model.input, output=predictions)
model_inception.summary()
```

This brought the validation loss to 0.44 and when the model was tested on the test set, the F1 score came out to be 0.8892.



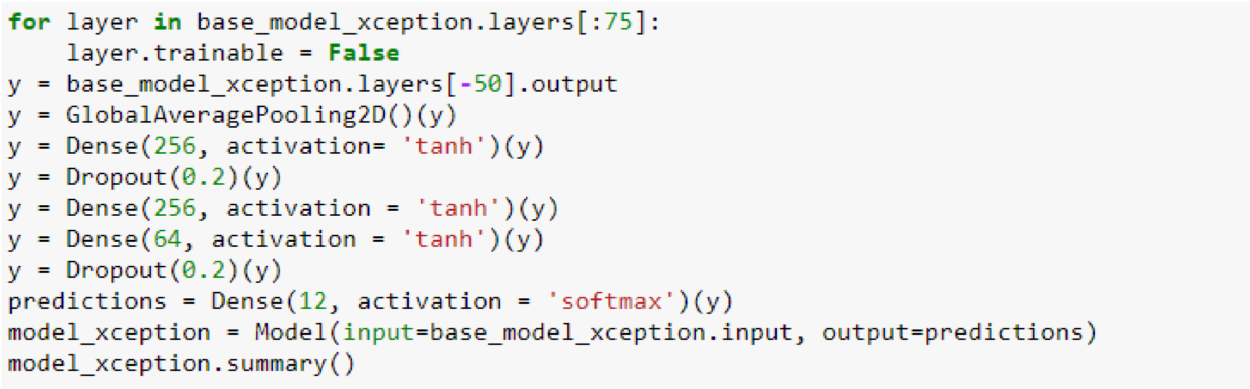
Evaluating the performance of the model - F1 score

```
inc_performance = evaluate_model(model_inception, 'saved_models/weights.best.from_tlInception.hdf5', test_tensors, y_test)
print('Performance evaluated by f1 score for the model created using TL is: ', inc_performance)
```

```
Performance evaluated by f1 score for the model created using TL is:  0.88920056101
```

Next, I built a new CNN on Xception pretrained network. But with this one, initially I removed around 20-30 layers from end, and froze the first 30-50 layers, added few fully connected layers with 'relu' activation function. But this architecture did not give any promising results with validation loss dangling around 0.45 – 0.32.

Hence, I increased the number of layers to freeze to 75, removed 50 layers from the end, changed the activation function on the fully connected layer to 'tanh' and retrained the model. I did notice improvement in the performance but also noticed that the model was trying to over-fit the training set, hence I added few Dropout layers in between the fully connected layer, and could see that model was performing well which brought the validation loss down to 0.27. Following is the outline of the final model using Xception:



```
for layer in base_model_xception.layers[:75]:
    layer.trainable = False
y = base_model_xception.layers[-50].output
y = GlobalAveragePooling2D()(y)
y = Dense(256, activation= 'tanh')(y)
y = Dropout(0.2)(y)
y = Dense(256, activation = 'tanh')(y)
y = Dense(64, activation = 'tanh')(y)
y = Dropout(0.2)(y)
predictions = Dense(12, activation = 'softmax')(y)
model_xception = Model(input=base_model_xception.input, output=predictions)
model_xception.summary()
```

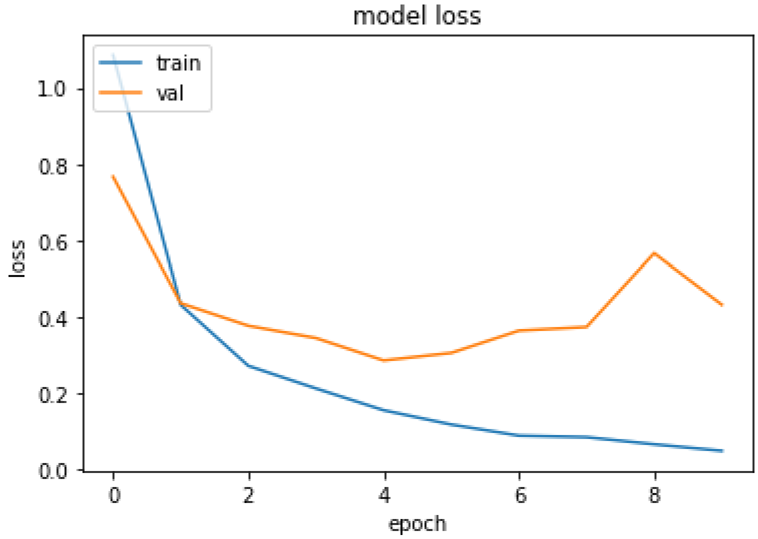
With this set up, the F1 score of model tested on the test set came out to be 0.9018. I believe this is good result considering I'm using the dataset as-is without any augmentation or masking.

IV. Results

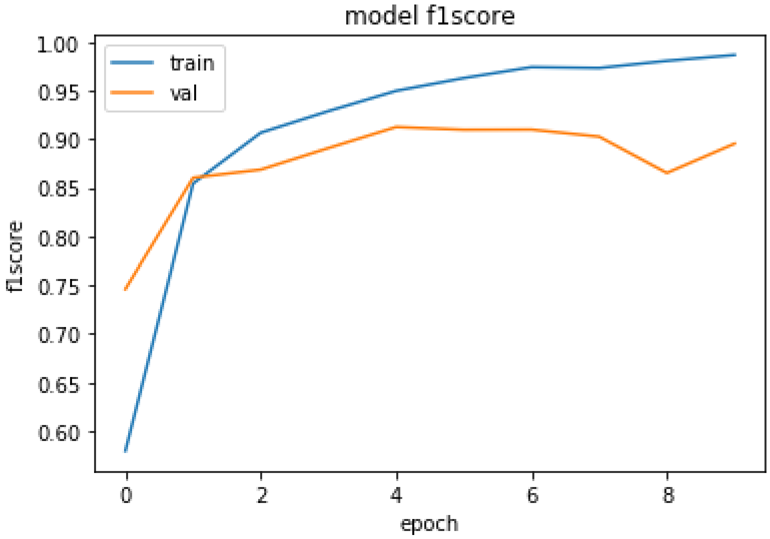
Model Evaluation and Validation

Though the model built using the InceptionV3 pre-trained network performed significantly better than my benchmark model, its performance is not better than the one built using the Xception pre-trained network and hence the latter one is my final classifier choice for this problem.

The model built using the Xception pre-trained network showed low training loss ~ 0.04 and ~ 0.28 validation loss. Also the F1scores for training set and validation set came out to be ~0.98 and ~0.91 respectively. On plotting the training and validation loss, we observe that although loss for the training set keep diminishing with the number of epochs, the validation set does not show any significant improvement after 4th epoch



Similar observation is noted in the plot between model's F1 score on the training and validation set. The training set appears to reach the ideal F1 score, whereas the validation set reaches its highest near 4th epoch and then reaches a plateau.



On testing the model’s performance on the unseen dataset – the test dataset, with the weights obtained from the training process, the classifier scored an F1 score of 0.9018. 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.

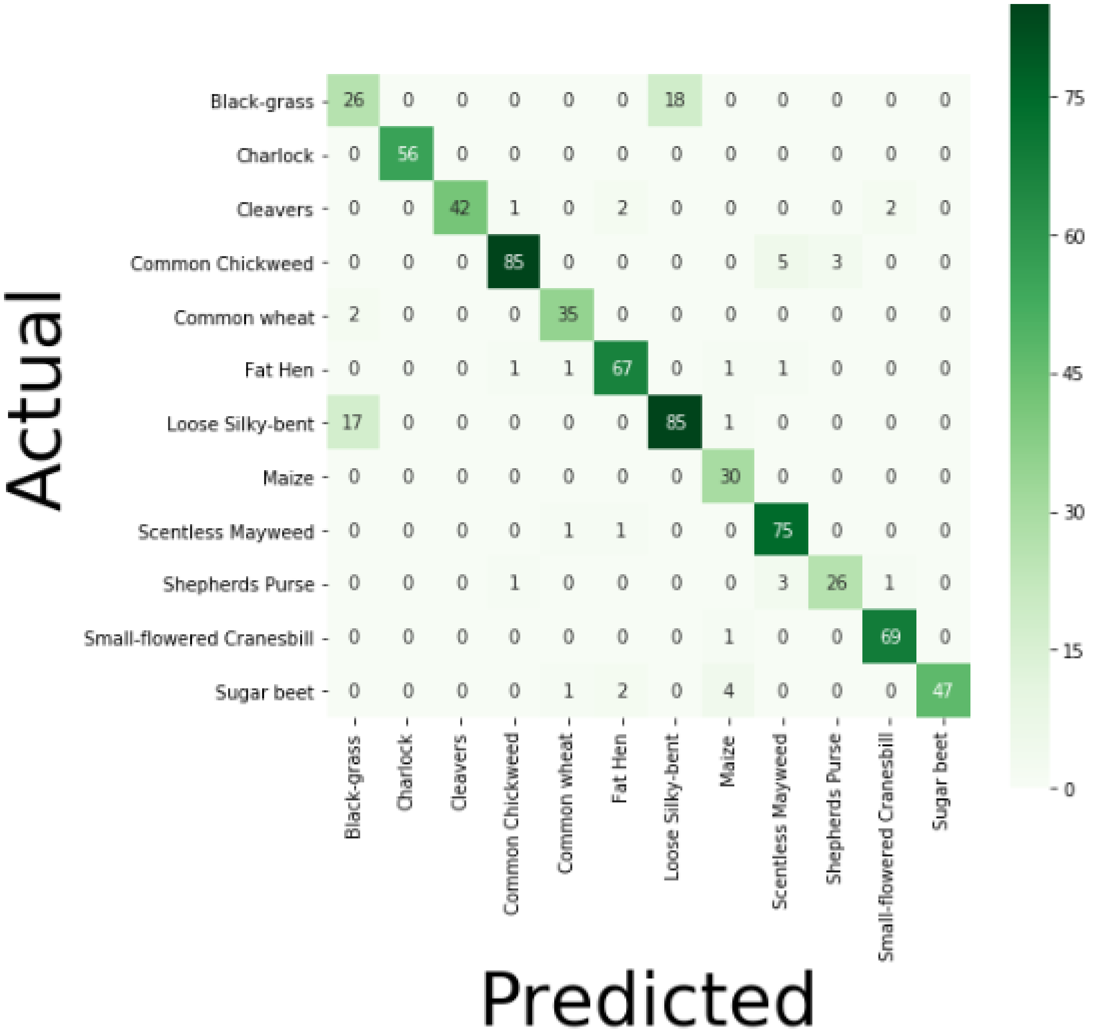
Justification

The results of the final classifier are much better than that of the benchmark model. An F1 score of ~0.9018 is a decent score when compared to ~0.68 (benchmark model’s performance). I believe the final solution will definitely contribute significantly towards solving the current problem and also with more training data there are possibilities of improving the model further.

V. Conclusion

Free-Form Visualization

I plotted the confusion matrix to observe which category is poorly classified by the classifier and observe its performance visually. We can see in the below confusion matrix plot, that the major misclassification happened between Loose Silky-bent and Black-grass. It looks like the classifier is having difficulty classifying these two categories. 18 of 44 Black-grass samples are classified as ‘Loose Silky-bent’ and similarly 17 of 102 Loose Silky-bent samples are classified as ‘Black-grass’. Hence, this is where the classifier needs improvement as these misclassification amount to 50% of the misclassification overall. I believe overcoming this challenge will boost the F1-score significantly.



Reflection

I wanted to pick a problem that would give me much more clarity and exposure on how to deal with image classification problem, and I believe that choice of this problem has justified that need.

For this problem, after downloading the dataset from Kaggle, I loaded the dataset using scikit learn's load_files() function and converted the categorical file names into array of 1s and 0s using the one-hot encoding. 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 in terms of number of color channels, background, size of each image etc. After splitting the dataset into train, validation and test sets, the images were converted into 4D tensors for further processing. Once I built CNN models, I fed these tensors and evaluated the model's performance using the F1-score metrics. I learnt, over multiple iterations, how does the number of layers in the convolutional network have an impact on the performance of the classifier, how does adding dropout layers reduce potential overfitting, what proportion of layers should be frozen while using a pre-trained network and what proportion should be removed according to the problem and dataset at hand. 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.

It is satisfying to see that the final model performs exceptionally well than the model built from scratch (which was set as my benchmark model).

Improvement

As an improvement and future work, I would like to try data masking on the training set. Noise from the background of the images can be cancelled by masking images. I believe that without the background noise and restricting the visibility to the green leaves, the model can be trained better, and we may notice 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 significant improvement in the performance.