Machine Learning Engineer Nanodegree

CAPSTONE PROJECT
Seedling Classification

Pawan Bhandarkar
May 17, 2019

DEFINITION

Project Overview

Can you differentiate between a crop and a weed?

In the country of India, the answer to this question could mean the difference between life and death for a population of about 500 million farmers and agricultural labourers spread across the rural regions. With the advent of technology, it is up to us, the Data Scientists and Engineers of this country to use our skills to help make their lives easier.

In this project, I have developed a model that would help distinguish between 12 different types of seedlings based on image. By knowing the type of seedling, we can come to a decision of whether to let it grow or to remove it. The model is build using Transfer Learning, based on a pre-trained Xception model and with a sufficient amount of pre-processing is applied to the data, the model is able to achieve significant results.

Problem Statement

The problem statement is acquired from the kaggle competition which was hosted a year ago : https://www.kaggle.com/c/plant-seedlings-classification and the goal is to answer the question that this project report started with: Can you differentiate between a crop and a weed? My strategy for solving this problem can be summed up in the following steps:

1) Performing data exploration to understand the nature of the data
2) Identify and apply the necessary pre processing steps to the data
3) Develop a benchmark model using a CNN built from scratch
4) Use Transfer Learning to create the final model
5) Train, test and evaluate the final model's performance

The final model is expected to be able to perform reasonably well at distinguishing between different seedlings so that it may be applied in the real world to aid farmers.

Metrics

The Kaggle competition description describes using the mean F-score as the metric for the evaluation and that is what I will also be used in this project. So it's worthwhile to know what the F Score really is. The traditional F-measure or balanced F-score (F 1 score) is the harmonic mean of precision and recall:

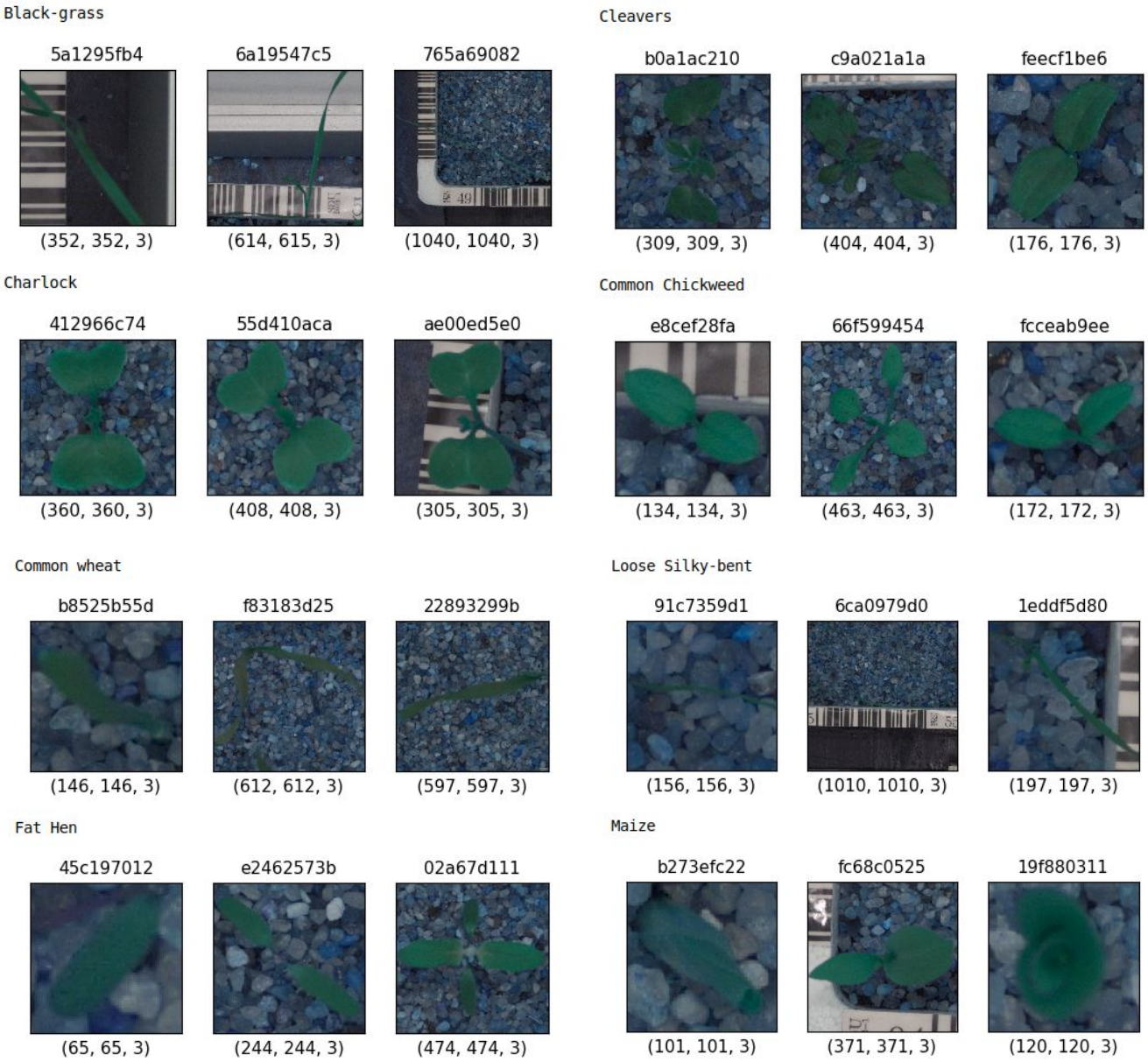
$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

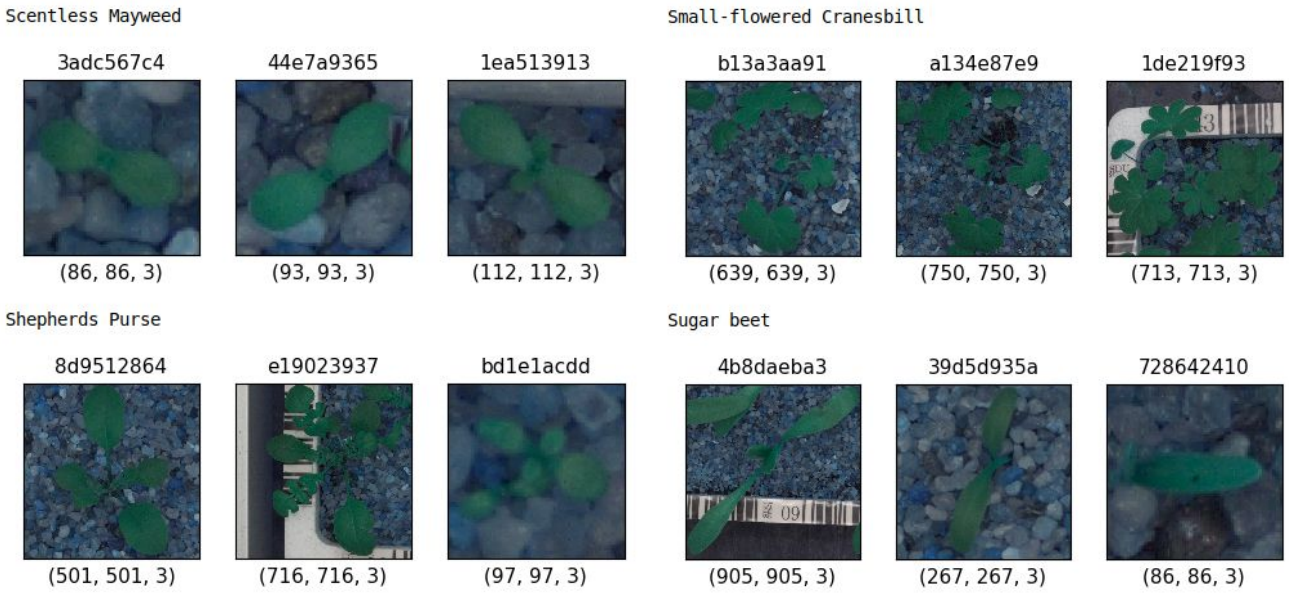
The same evaluation metric will be used to evaluate both the benchmark as well as the final models and a graph will be used to visualize the results.

ANALYSIS

Data Exploration

The dataset for this project is downloaded from the one hosted on the Kaggle competition page mentioned above. Right off the bat, we can see that the data is organised into folders, with each folder being named after a class i.e, one of the 12 different plant seedling identifiers. The data itself is present as PNG images within each folder. Upon inspection we can see that not all images are of the same size and this poses a problem to our project. A Convolutional neural network expects that all data should belong to the same input space, i.e, have the same number of features to be trained on.

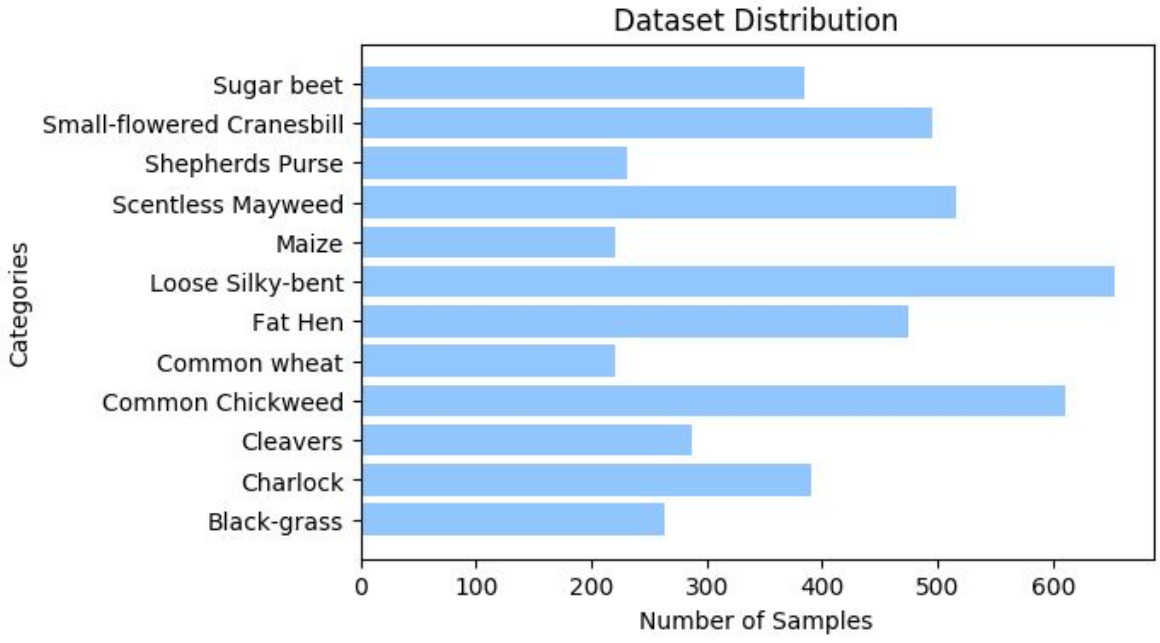




Another issue that we can identify instantly from the above samples of the data is the amount of noise present in it. While our analysis based solely on what the plant looks like, we do not care about the background i.e, the rocks and the mud underneath the plants. We shall address these issue in the data pre processing section.

Exploratory Visualization

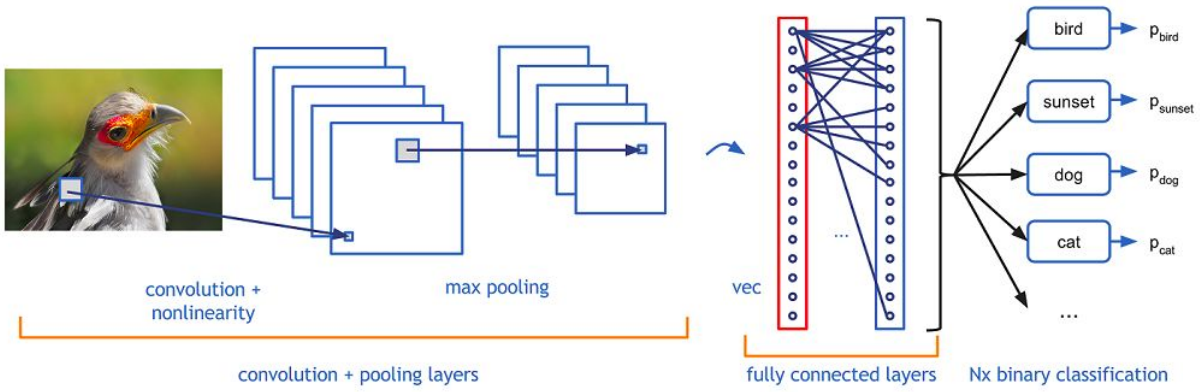
The following graph shows the distribution of samples in the data set. We can clearly see that the data set is imbalanced with Loose Silky-bent having the most samples.



Some approaches to solve this include oversampling and data augmentation. But after some preliminary benchmarks, it was found that we were able to achieve quite satisfactory results without having to augment the data data so taking into account, the time available for the project completion, I decided to stick with the data on hand.

Algorithms and Techniques

The problem at hand is one of image classification and the Machine Learning Algorithm best suited for tackling such problem are Convolutional Neural Networks. The following image sums up the working of a CNN. It takes as input a set of matrices corresponding to an image and outputs a class that it predicts the image belongs to.

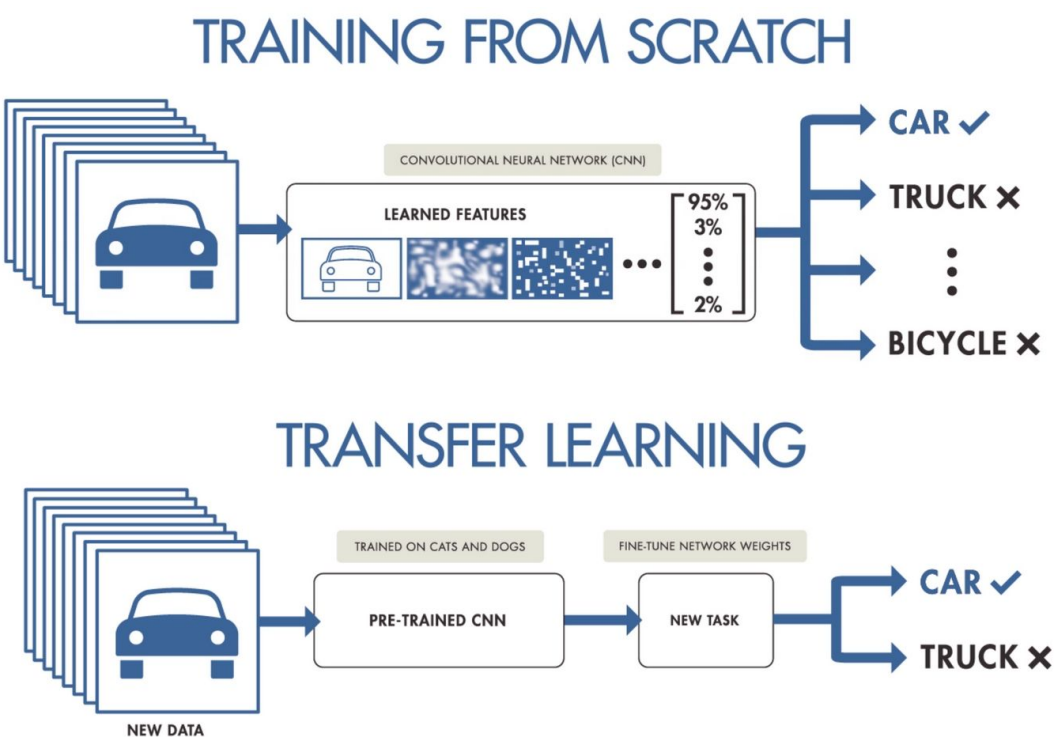


[Image obtained from this github post by Adit Deshpande]

While it is possible to construct a CNN from scratch and use that for our problem statement, generally it is preferable to build a model using the concept of Transfer Learning, where we take a pre-trained keras model, trained on say the Imagenet database and the modify and fine tune the final few layers to suit our classification task.

Then by training only the last few layers, we can achieve much better results while simultaneously saving on time and training costs. This approach was used in the Dog Breed Classifier project i worked on for this course and it delivered phenomenal results.

The following image sums up how transfer learning works:



[Image obtained from Pinterest]

Benchmark

For this project, I have two different benchmarks implemented, as I had explained in the proposal for this project:

1) A random chance model that tries to randomly guess the class of the image
2) A CNN model built from scratch and trained on the processed data

These two benchmarks can then be evaluated using the same metric i.e, the F1 score and then compared to the final model. This would give us a clear cut understanding of how much better the final model is, compared to the benchmarks. The CNN used to benchmark (2) was built using the keras library and comprises of Convolutional, Max pooling and Dense Layers and uses the relu activation function in the hidden layers. A GAP layer is also introduced just before the output.

A summary of the benchmark CNN is as follows:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_1 (MaxPooling2	(None, 111, 111, 16)	0
conv2d_2 (Conv2D)	(None, 109, 109, 32)	4640
max_pooling2d_2 (MaxPooling2	(None, 54, 54, 32)	0
conv2d_3 (Conv2D)	(None, 52, 52, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 26, 26, 64)	0
conv2d_4 (Conv2D)	(None, 24, 24, 128)	73856
max_pooling2d_4 (MaxPooling2	(None, 12, 12, 128)	0
global_average_pooling2d_1 (	(None, 128)	0
dense_1 (Dense)	(None, 12)	1548

Total params: 98,988
Trainable params: 98,988
Non-trainable params: 0

I trained the above model for a good 50 epochs, while also making use of check pointers and early stopping,, setting the patience to 5 and and delta of 0.05. (i.e, After 5 epochs of training with improvements in the validation loss being less than 0.05, stop the training process).

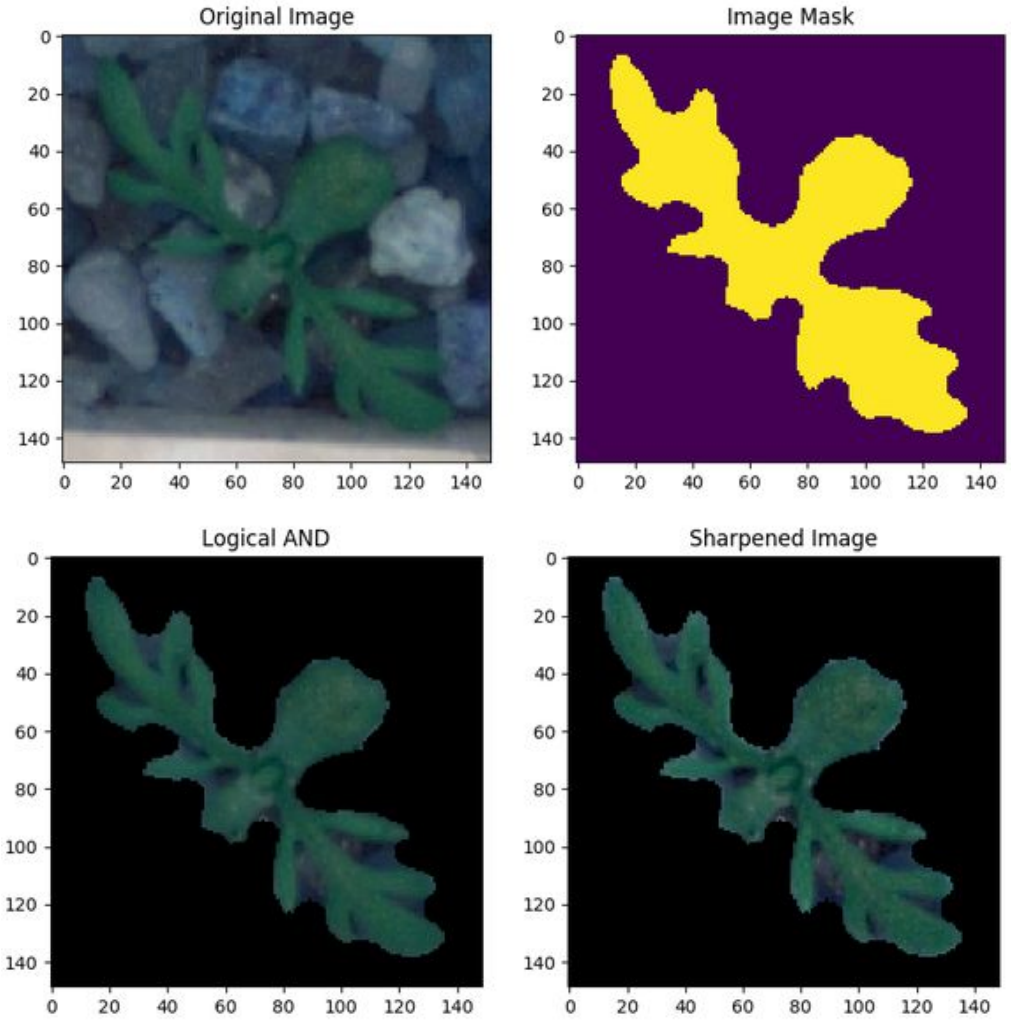
After the training process, I loaded the best weights and tested the benchmark model on the testing data and it achieved an F1 score of **0.856** after 35 epochs. That is the threshold that the final model has to beat.

METHODOLOGY

Data Preprocessing

The main issue we discussed in the data exploration section is the presence of background noise in the images. Again, we only care about the actual seedlings themselves and not the rocks or the mud underneath it. One way to get rid of the background is my using maks. Now at first, I thought I would write the code for this by myself from scratch, but poking around on the internet, I came across a set of well documented and lean code written by Gábor Vecsei, here.

It did a good job of segmenting and sharpening the images resulting in a data set that was devoid of all background noise. The following set of images show the results of the pre processing on a random sample image of a seedling:



The above transformation is applied to each and every image in the dataset and saved back into the folder, replacing the original images. The code performing this transformation is quite simple and efficient and as a result, all images found in the wild can be processed this way by simply calling the one function.

The other preprocessing steps that were performed are as follows:

1) Scaled all the images to a fixed size of (224, 224, 3)
2) One-hot encoded the class labels.
3) Split the data into train, validation and test sets.

Implementation

Metrics

Generally, when we deal with traditional machine learning algorithms as in the case as one built using scikit learn, we can use the metrics defined in the sklearn.metrics module. The metric decided upon for our project was F1 score and unfortunately, keras doesn't have any built in functions to calculate and monitor the F1 score during the training process. But fortunately, the keras team has uploaded the source code for most of the metrics here and I have used the functions for precision, recall and fbeta_score for my project, borrowed from that github repo.

Algorithm

As I had mentioned, the algorithm I decided to use for this project was one involving transfer learning and this approach posed two major challenges:

1) Which pre-trained keras model do I use?
2) How many layers of the pre-trained model to retrain?

I decided to try out multiple different models before I settled upon Xception as my choice. To make this decision, I trained all the models on the un-processed data and recorded the F1 scored I obtained from each one. I also decided to train each model twice, once freezing the layers of the base model while only training the output layer and once keeping all the parameters trainable. I obtained the following scores for the models that I tested:

Model	Trainable Layers	Validation F1 Score
Xception	Output only	0.81
	All	0.896
InceptionV3	Output only	0.74
	All	0.85
VGG16	Output only	0.83
	All	0.85

Out of the models that I tested, Xception proved to be the most promising one and since I had also observed similar results for the tests I had performed in my previous project, I decided to stick with this as the final model.

Refinement

Deciding upon the model was only the first step. After that came the second issue: deciding how many of the layers to keep frozen and how many to train. I tried a number of options and I discovered that the best results were obtained when keeping the first 100 layers of the Xception model frozen while training the rest. To the base model, I also added the following two layers:

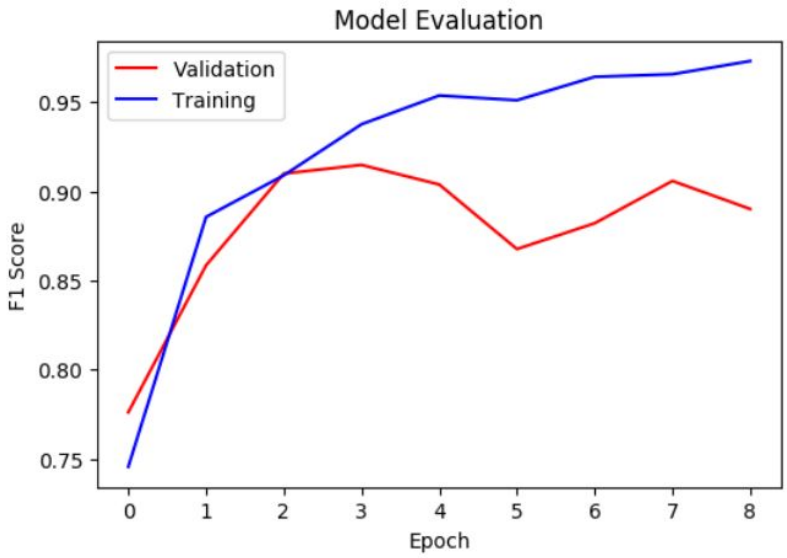
1) A Global Average Pooling layer (2D)
2) A Dense layer with a softmax activation function as the output layer

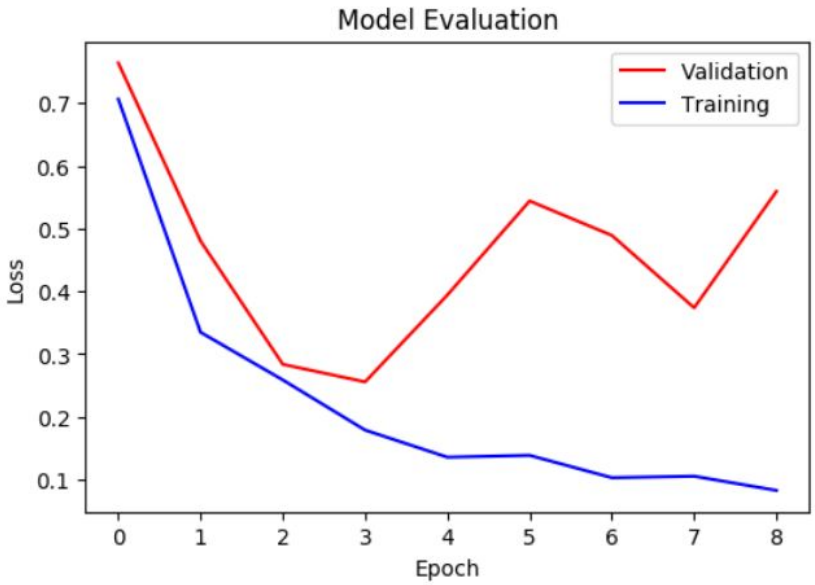
Another choice that I had to make was in the optimizer to be used. Out of the ones that I tried, the Adam optimizer was the one that gave me the best results so I decided to use that. Once all the above steps were done, I trained and tested the model on the processed and masked data, while freezing the first 100 layers and I observed a final result of 0.915 which in my opinion is a good result since I had to work with a reduced training set, accounting for the fraction that I had to set aside as testing and validation data. I strongly believe that If i had access to more data, I could have achieved better results.

RESULTS

Model Evaluation and Validation

I decided to go with the concepts that I had learned during my Machine Learning Foundation Nanodegree to evaluation and validate my models and that is, the validation curves. By using the history that keras automatically records and stores in its "history" attribute, I was able to access the f1 scores and loss values that were calculated for every epoch, for both the training set as well as the validation set. Then I plotted them using line graphs as follows:





From the above two graphs, we can see that there are clear signs of overfitting in the training process however, due to early stopping and because we use checkpointers to monitor the training, we save only the best weights from the entire training. i.e , the weights of the model that achieved the lowest validation loss and generally the highest F1 score. This is further fortified when we check the final model against the test set that we had initially kept aside.

Justification

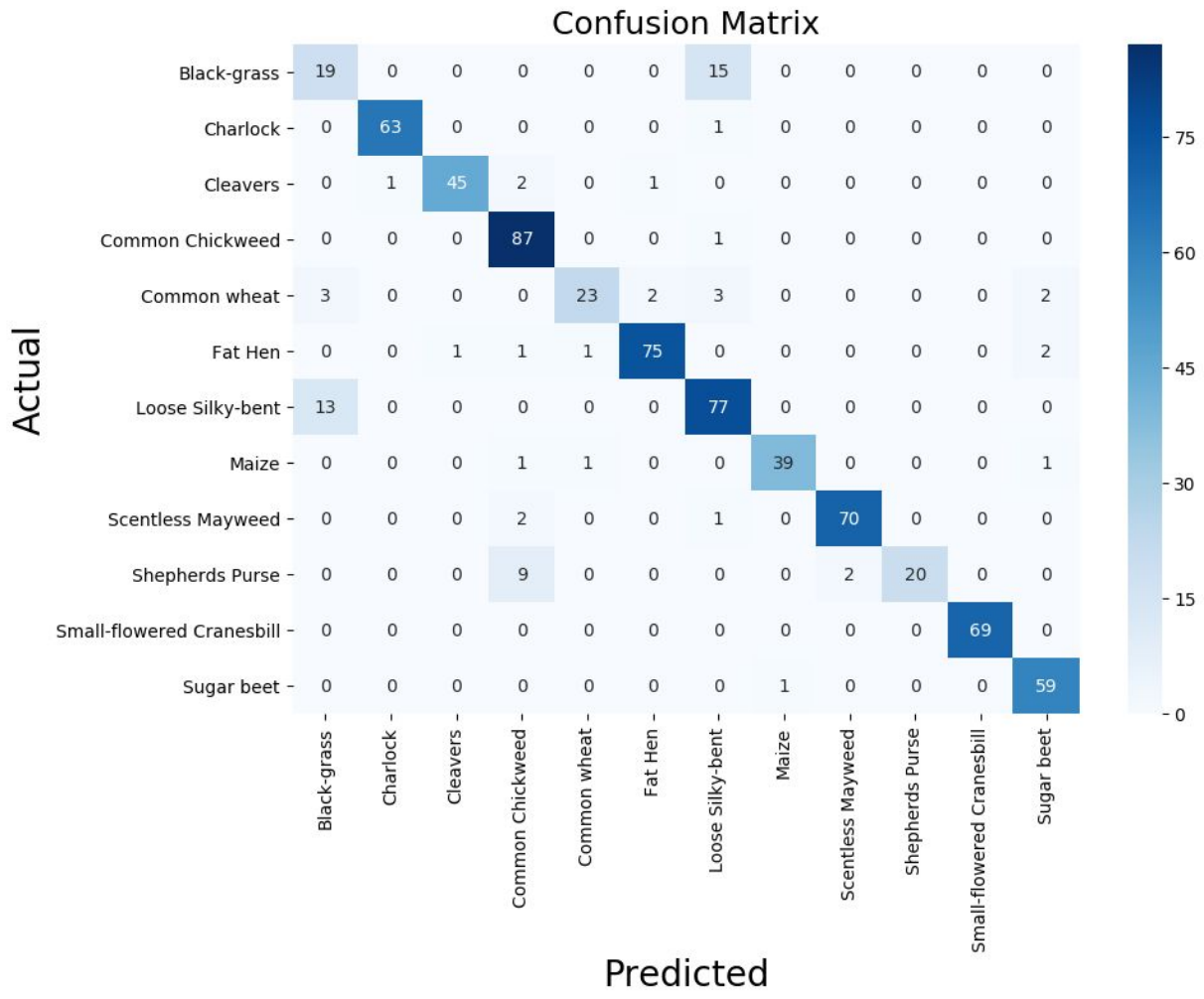
The final model achieved an F1 score of **0.915** which I would consider to be a significant enough improvement over the benchmark score of **0.856** which is approximately a 6% improvement over the benchmark model and in a country such as India, where agricultural is considered the “backbone” of the country, this different will be a significant one, in the long run.

However, I do believe that with a few improvements to the benchmark model, it might also be able to cross the 0.9 mark. But for the course of this project, I am quite satisfied with the improvement achieved through Transfer Learning.

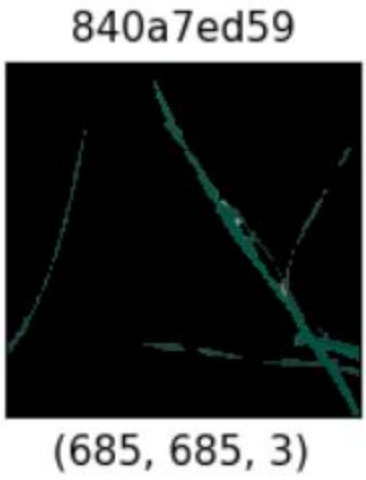
CONCLUSION

Free-Form Visualization

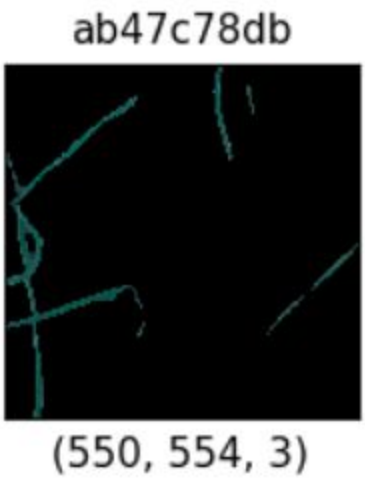
Here, as I had mentioned in the project proposal, I decided to use a seaborn heatmap to visualize the confusion matrix generated by running the final model on the testing set and the results are as follows:



We can see that the class that was the most difficult to predict correctly was Black grass. The reason for this, I suspect is that it has too few samples within the data set. Most of the misclassified samples of Black Grass were classified as Loose Silky Bent and a reason for this could be that the latter has the most number of samples and some of them might be similar to Black Grass, as we can see in the following samples:



Black Grass



Loose Silky-bent

Reflection

All in all, this project has been very fulfilling to work on. I chose this project out of sheer interest in the domain of image classification and given how important agriculture is to my people, I decided that this would be a suitable project to work on. From the start of this project, till its conclusion, I have learnt a lot. All the projects that I had worked on so far had templates and questions provided to us to follow along with questions to be answered. But this project felt quite liberating in that I was able to fully explore the project domain on my own and learnt how to tackle a real-world problem in the process.

Throughout this project, I worked on applying all that I had learnt in my Nanodegree and outside as well and it taught me not only how to apply what I already know but also how to look up functions, read documentation and develop lean code on my own.

The most challenging aspect for me, in this project was actually getting access to a GPU instance on AWS. So far, I had been using the workspaces provided by udacity and for my other projects, my laptop's CPU would suffice. But for this project, I decided to go with a p2.xlarge instance from AWS and thanks to tensorflow's GPU support and CUDA being installed on the AWS instance, I was able to quickly and reliably train my Neural Networks.

Improvement

There are definitely some areas where I think I would have been able to improve on the model given the time and the resources to do so and they are as follows:

1) As I had explained in the data exploration part of this report, the model would benefit greatly by increasing the amount of data available for training the mode.
2) Image Augmentation could be used to improve on the training scores.
3) Oversampling could be used to solve the problem of the imbalanced dataset that we discussed in the visualization section of this report.

Applying these improvements might help us improve the final score of the model and push it toward the scores that we see on the Kaggle leaderboards for this competition.