Machine Learning Engineer Nanodegree

Capstone Project

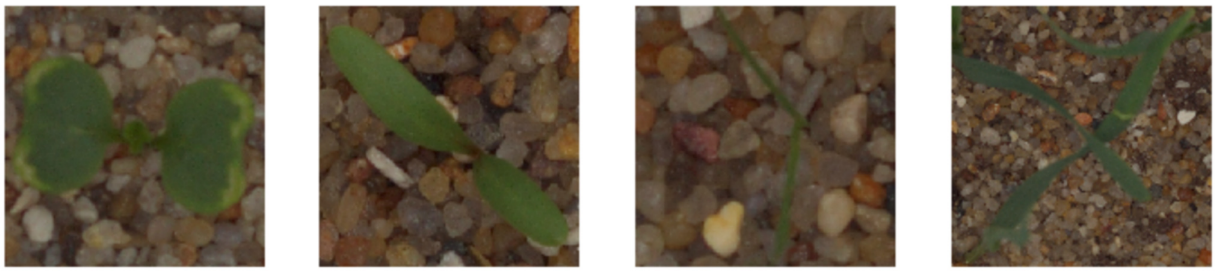
By Narendra Kumar Muthum
December 17th, 2018

I. Definition

Project Overview

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



Rapid and accurate identification of weeds at the seedling stage is the first step in the design of a successful weed management program that saves producers and land managers time, money, and reduces herbicide use. How does weed seedling identification provide these benefits? First, weed management is typically much easier, less costly, and more effective at the seedling or juvenile (e.g. rosette) stage than on mature plants. Second, controlling a weed during early growth stages allows desirable neighboring vegetation to grow better, thereby improving overall plant community vigor. Finally, improper identification can result in misapplication of a management tactic such as herbicides or failure to adequately control the weedy plant species at the time that it is most vulnerable. Once a species has been correctly identified, an Integrated Weed Management can be designed that combines the use of biological, cultural, mechanical, and chemical practices to manage weeds. The main goals of an Integrated Weed Management program are to:

♣ use preventive tools to maintain the crop or desired vegetation and limit weed density to a tolerable, non-harmful level,

♣ avoid shifts in the composition of plant communities towards other weeds that may be even more difficult to control,

♣ develop sustainable management systems that maximize environmental quality, productivity, and revenues. Thus, designing a successful Integrated Weed Management (IWM) program requires an understanding of the biological and ecological factors that influence the growth and development of weeds. Part of this understanding is the need to correctly identify all different kinds of weed species.

To automate this process Aarhus University group partner with University of Southern Denmark are hosting this dataset as a Kaggle competition in order to give it wider exposure, to give the community an opportunity to experiment with different image recognition techniques, as well to provide a place to cross-pollenate ideas.

Problem Statement

Determine the species of a seedling from an image Species of a seedling from an images labeled by Aarhus University Signal Processing group by identifying 12 species of objects in the image such as '`Black-grass`', '`Charlock`', '`Cleavers`', '`Common Chickweed`'... along with sand, stones and bar codes.

The goal is to determine or predict the likelihood that a species is from a certain class from the provided classes, thus making it a multi-class classification problem in machine learning terms.

These twelve target species classes are provided in train dataset. The goal is to train a CNN that would be able to classify fishes into these twelve classes.

To quantifiable or measures the model by submissions are evaluated on Mean F-Score, which at Kaggle is actually a micro-averaged F1-score.

Metrics

The metric used for this Kaggle competition is **multi-class logarithmic loss** (also known as categorical cross entropy)

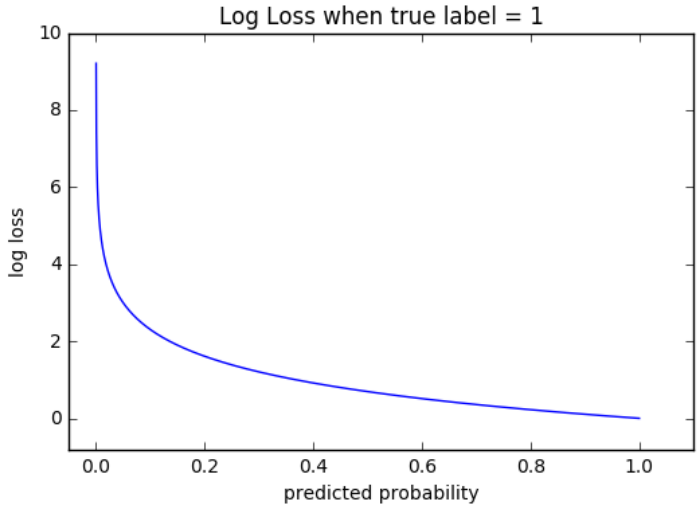
By selecting categorical cross entropy as our loss function since the categorical cross entropy is preferred for mutually-exclusive multi-class classification task (where each example belongs to a single class) compared to other metrics. For each image, we must
submit a set of predicted probabilities (one for every image). The formula of log loss is
then,

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

where N is the number of samples or instances, M is the number of possible labels, yij is a binary indicator of whether or not label j is the correct classification for instance i, and pij is the model probability of assigning label j to instance i. A perfect classifier would have a Log Loss of precisely zero. Less ideal classifiers have progressively larger values of Log Loss.

Looking Closer

Let's take a closer look at this relationship. The plot below shows the Log Loss contribution from a single positive instance where the predicted probability ranges from 0 (the completely wrong prediction) to 1 (the correct prediction). It's apparent from the gentle downward slope towards the right that the Log Loss gradually declines as the predicted probability improves. Moving in the opposite direction though, the Log Loss ramps up very rapidly as the predicted probability approaches 0. That's the twist I mentioned earlier.



Using the whole dataset as the baseline dataset for model evaluation because the Plant Seeding dataset is pretty small. The baseline model of our project is CNN with VGG 16 and multi-class Logistic regression. Hence chosen micro-averaged F1 score as the major evaluation matrix since it is selected in the Kaggle competition, and it is easier to compare the performance of our model with previous works. In addition, F1 score could balance the precision and recall and yield a more realistic indication of model performance. Similarly, we can use confusion matrix to visualize the prediction results.

Besides baseline models, it is obvious to experiment with models such as simple Neural Network model, CNN models (pre-trained models) and ensemble model to see their performances. The state of the art F1 score is achieved by a Xception, VGG 16, InceptionV3 models.

II. Analysis

Data Exploration

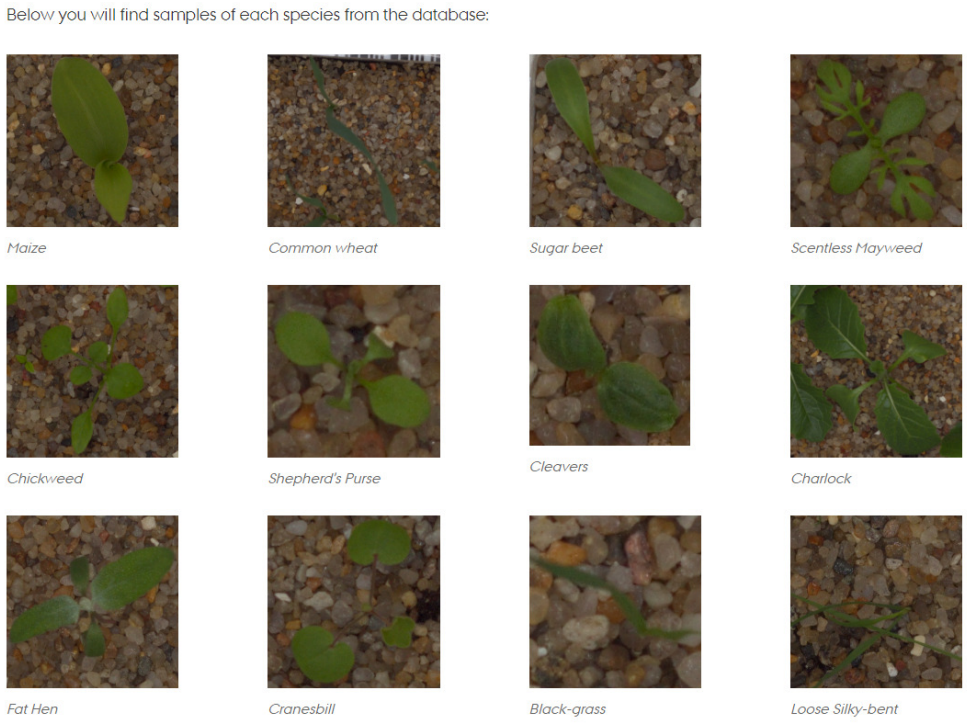
By extending my appreciation to the Aarhus University Department of Engineering Signal Processing Group for hosting the original data. The dataset is available at https: // vision. eng. au. dk/ plant-seedlings-dataset.

They have provided with training set 4750 labeled images (1.73GB) and a test set 794 images (91MB) of plant seedlings at various stages of grown without label. 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 distribution is as follows:

Species	Total Images
Loose Silky-bent	654
Common Chickweed	611
Scentless Mayweed	516
Small-flowered Cranesbill	496
Fat Hen	475
Charlock	390
Sugar beet	385
Cleavers	287
Black-grass	263
Shepherds Purse	231
Common wheat	221
Maize	221

By seeing above species summary and distribution, we can assure that provided training species classes data samples are balanced in nature. Now we can use the train_test_split function in order to make the split into training data and validate data. The training set contains a known output and the model learns on this data to generalize this to other data later on. We will have the validate dataset (or subset) to test our model's prediction on this subset. It's usually around 80/20 or 70/30.

Sample plant seeding images shown below along with species class.



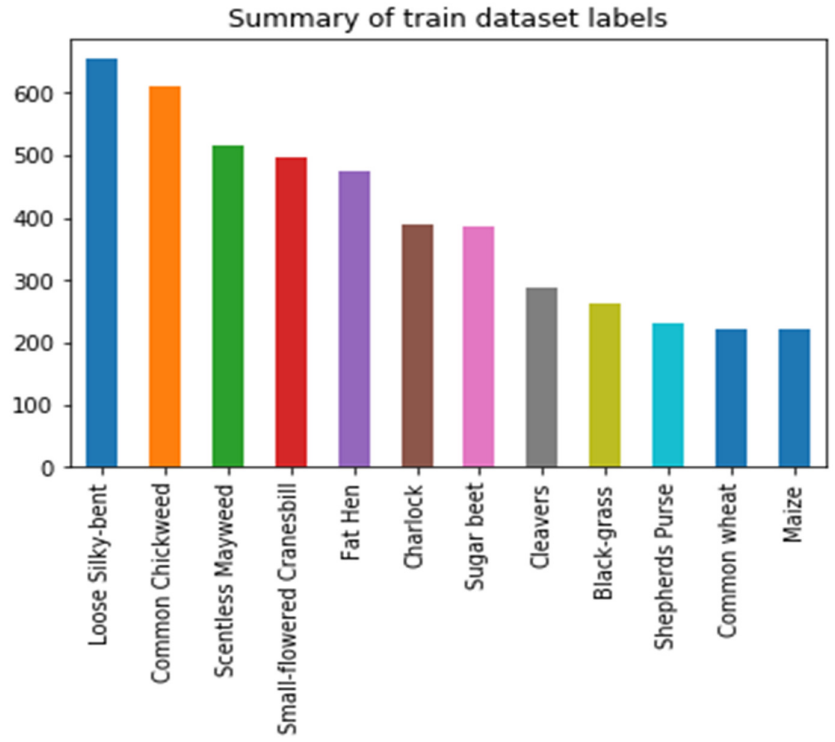
To conclude, the end result is for each file in the test set, we must predict a probability for the species variable. The file should contain a header and have the following format (in CSV file format):

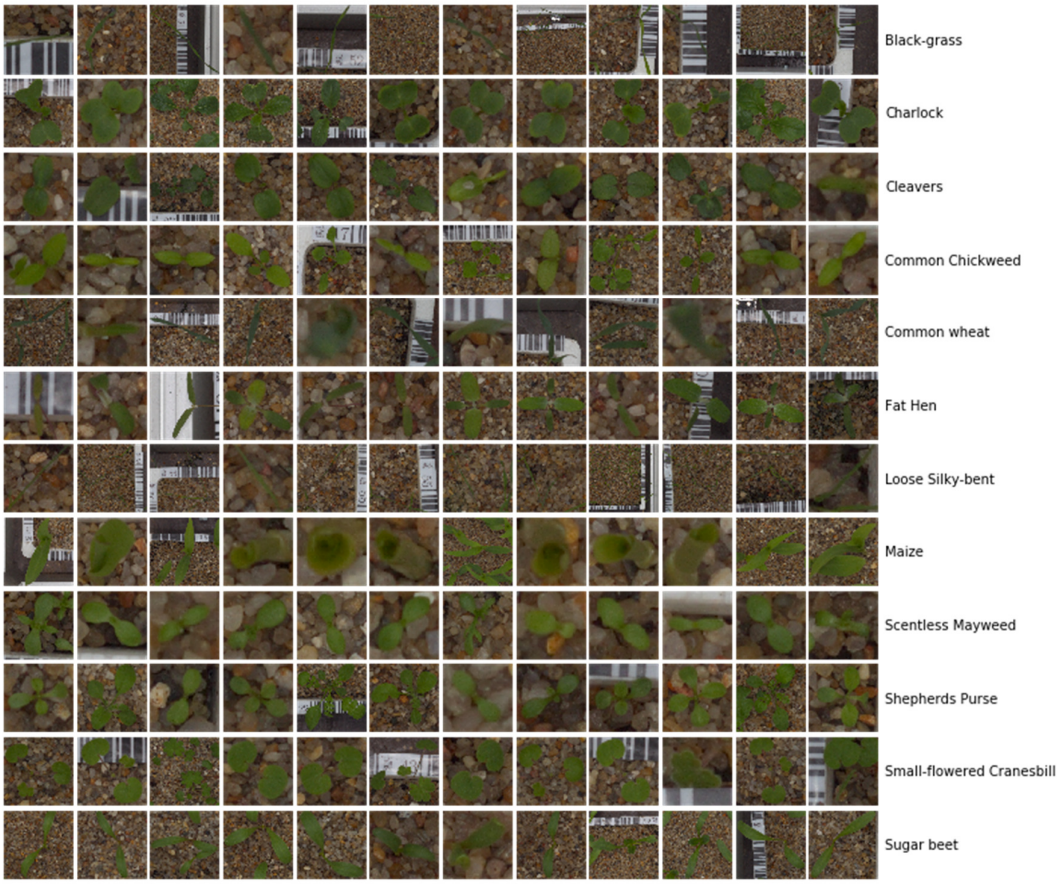
```
file, species
0021e90e4.png, Maize
003d61042.png, Sugar beet
```

007b3da8b.png, Common wheat
etc.

Exploratory Visualization

For understanding train data image distribution whether our data is balanced or unbalanced classes in image classification problem. The count of images for each class in training dataset is obtained and the below species summary and distribution clearly shows, and we can assure that provided training species classes data samples are balanced in nature. Also trying to show in below all these twelve classes with 12 images in a single shot.





Algorithms and Techniques

Transfer Learning:

Transfer learning refers to the process of using the weights a pretrained network trained on a large dataset applied to a different dataset (either as a feature extractor or by finetuning the network). Finetuning refers to the process of training the last few or more layers of the pretrained network on the new dataset to adjust the weight. Transfer learning is very popular in practice as collecting data is often costly and training a large network is computationally expensive. Here weights from a convolutional neural network pretrained on imagenet dataset is finetuned to classify plant seeds.

What is a Pre-trained Model?

A pre-trained model has been previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on. Learned features are often transferable to different data. For example, a model trained on a large dataset of bird images will contain learned features like edges or horizontal lines that you would be transferable your dataset.

Why use a Pre-trained Model?

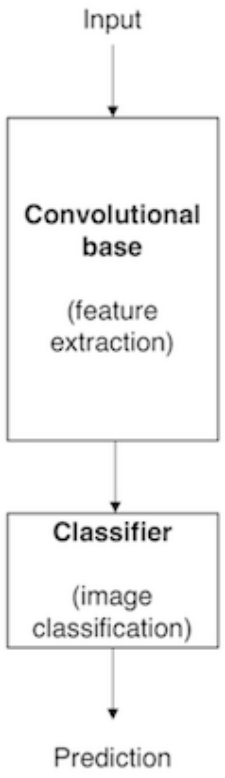
Pre-trained models are beneficial to us for many reasons. By using a pre-trained model you are saving time. Someone else has already spent the time and compute resources to learn a lot of features and your model will likely benefit from it.

A typical CNN has two parts:

Convolutional base, which is composed by a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image. For an intuitive explanation of convolutional and pooling layers, please refer to Chollet (2017).

Classifier, which is usually composed by fully connected layers. The main goal of the classifier is to classify the image based on the detected features. A fully connected layer is a layer whose neurons have full connections to all activation in the previous layer.

Here is our model process approach looks like:



Benchmark method:

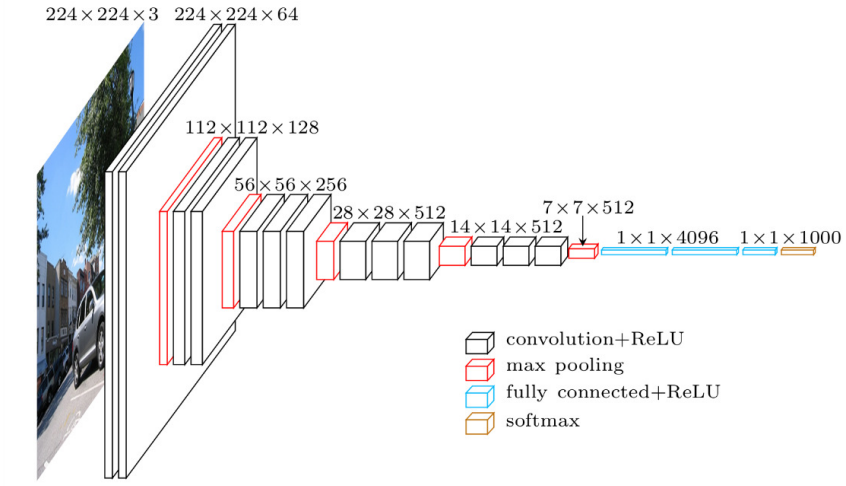
Logistic Regression : Logistic regression is one of the most popular supervised classification algorithm. This classification algorithm mostly used for solving binary classification problems. People follow the myth that logistic regression is only useful for the binary classification problems. Which is not true. Logistic regression algorithm can also use to solve the multi-classification problems. In **multi-classification** problems given the dimensional information of the object, Identifying the shape of the object. In the multi-classification problem, the idea is to use the training dataset to come up with any classification algorithm. Later use the trained classifier to

predict the target out of more than 2 possible outcomes. When it comes to the multinomial logistic regression the function is the **Softmax Function**.

To standardise the evaluation of classification results obtained with the database, a benchmark based on f1 scores is proposed. So, we will pick pretrained VGG-16 model with simple logistic regression as a benchmark and try to beat the benchmark with hyperparameter turning. We will also try Ensemble methods if the hyperparameter tuning does not improve the score.

A well-designed convolutional neural network should be able to beat the random choice baseline model easily considering even the simple logistic regression model clearly surpasses the initial benchmark. However, due to computational costs, it may not be possible to run the transfer learning model with VGG-16 architecture for sufficient number of epochs so that it may be able to converge.

VGG(16) Architecture: Winner of the ImageNet ILSVRC-2014 competition, VGGNet was invented by Oxford's Visual Geometry Group , The VGG architecture is composed only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier. The pretrained model is available in Caffe, Torch, Keras, Tensorflow and many other popular DL libraries for public use.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

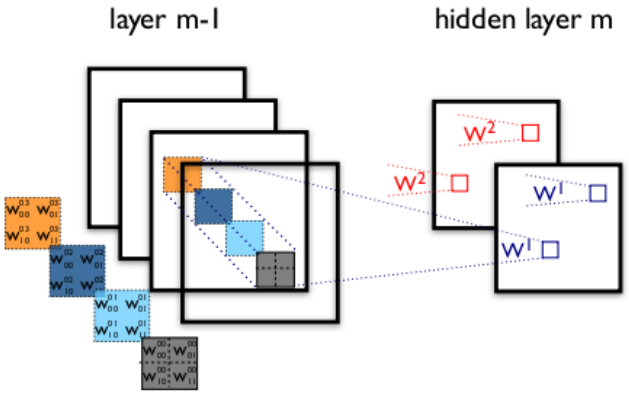
Layers: VGGNet consists of 16 convolutional layers and is very appealing because of its very uniform architecture. It only performs 3×33\times 33×3 convolutions and 2×22\times 22×2 pooling all the way through. It is currently the most preferred choice in the community for extracting features from images. However, VGG Net consists of 140 million parameters, which can be a bit challenging to handle our limited small images samples.

Xception Architecture: Xception V1 model, with weights pre-trained on ImageNet. On ImageNet, this model gets to a top-1 validation accuracy of 0.790 and a top-5 validation accuracy of 0.945. Note that this model only supports the data format 'channels_last' (height, width, channels). The default input size for this model is 299x299.

Xception stands for "extreme inception." Rather like our previous VGG-1 architecture, it reframes the way we look at neural nets—conv nets in particular. And, as the name suggests, it takes the principles of Inception to an extreme.

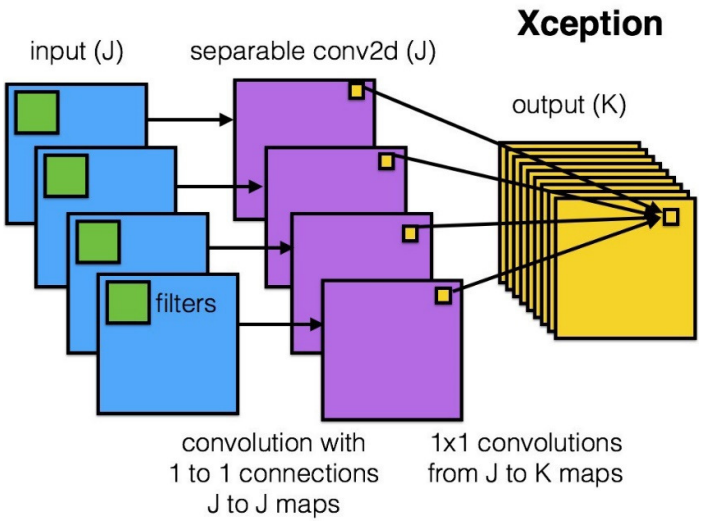
Here's the hypothesis: "*cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly*."

What does this mean? Well, in a traditional conv net, convolutional layers seek out correlations across both *space* and *depth*. Let's take another look at our standard convolutional layer:



In the image above, the filter simultaneously considers a spatial dimension (each 2×2 colored square) and a cross-channel or "depth" dimension (the stack of four squares). At the input layer of an image, this is equivalent to a convolutional filter looking at a 2×2 patch of pixels across all three RGB channels. Here's the question: is there any reason we need to consider both the image region and the channels at the same time?

In Inception, we began separating the two slightly. We used 1×1 convolutions to project the original input into several separate, smaller input spaces, and from each of those input spaces we used a different type of filter to transform those smaller 3D blocks of data. Xception takes this one step further. Instead of partitioning input data into several compressed chunks, it maps the spatial correlations for *each output channel separately*, and then performs a 1×1 depthwise convolution to capture cross-channel correlation.



The author notes that this is essentially equivalent to an existing operation known as a "depthwise separable convolution," which consists of a depthwise convolution (a spatial convolution performed independently for each channel) followed by a pointwise convolution (a 1×1 convolution across channels). We can think of this as looking for correlations across a 2D space first, followed by looking for correlations across a 1D space. Intuitively, this 2D + 1D mapping is easier to learn than a full 3D mapping.

And it works! Xception slightly outperforms Inception v3 on the ImageNet dataset, and vastly outperforms it on a larger image classification dataset with 17,000 classes. Most importantly, it has the same number of model parameters as Inception, implying a greater computational efficiency. Xception is much newer (it came out in April 2017), but as mentioned above, its architecture is already powering Google's mobile vision applications through MobileNet.

In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. "Xception" means "Extreme Inception", as this new model uses depthwise separable convolutions, which are at one extreme of the spectrum described above.

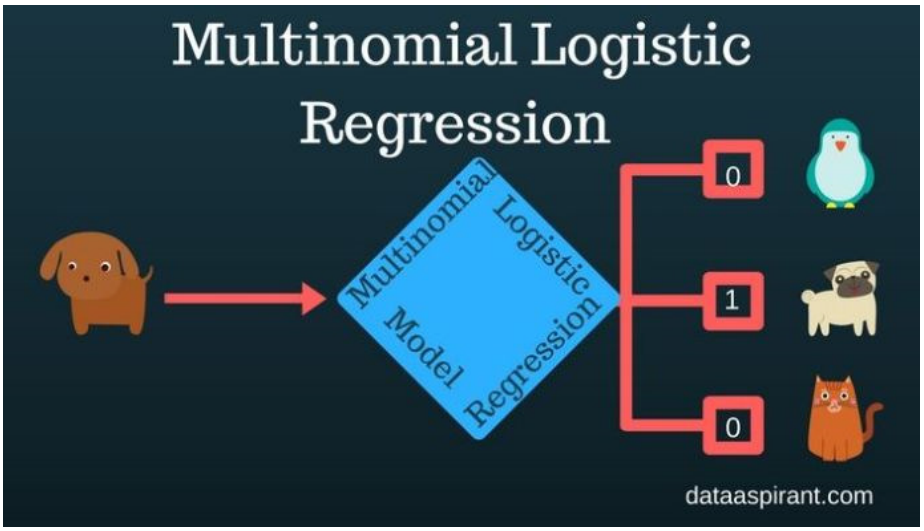
Fun facts: The author of Xception is also the author of Keras. Francois Chollet is a living god.

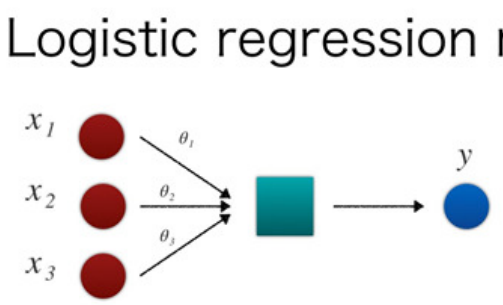
Unsupervised learning Model:

Multinomial Logistic Regression: In the pool of supervised classification algorithms, the logistic regression model is the first most algorithm to play with. This classification algorithm is again categorized into different categories. These categories are purely based on the number of target classes.

If the logistic regression model used for addressing the binary classification kind of problems it's known as the binary logistic regression classifier. Whereas the logistic regression model used for multiclassification kind of problems, it's called the multinomial logistic regression classifier.

In statistics, multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes. That is, it is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables (which may be real-valued, binary-valued, categorical-valued, etc.).





In short:

- **Sigmoid function:** used in the logistic regression model for binary classification.
- **Softmax function:** used in the logistic regression model for multiclassification.

Softmax: softmax(x)

For a vector $x \in \mathbb{R}^n$, the **softmax** function is defined as

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Each element in softmax(x) is squashed to the range [0,1, and the sum of the elements is 1. Thus, the softmax function is useful for converting an arbitrary vector of real numbers into a discrete probability distribution.

Data Augmentation : Data augmentation is a regularization technique where we produce more images from the training data provided with random jitter, crop, rotate, reflect, scaling etc to change the pixels while keeping the labels intact. CNNs generally perform better with more data as it prevents overfitting.

Batch Normalization : Batch Normalization is a A recently developed technique by Ioffe and Szegedy which tries to properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. In practice we put the Batchnorm layers right after Dense or convolutional layers. Networks that use Batch Normalization are significantly more robust to bad initialization. Because normalization greatly reduces the ability of a small number of outlying inputs to over-influence the training, it also tends to reduce overfitting. Additionally, batch normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the network itself.

Coding was carried out using Python and Keras with Tensorflow as the backend. During the training process huge memory usage was observed when all the images were loaded and pre-processed. As a result this step was executed multiple times by changing the capacity of the compute instances to account for huge RAM.

Also during the transfer learning process, obtaining bottleneck features took time. Although GPU instances from cloud were used, training process during fine tuning took huge time(~ 6 hours). This may be due to the usage of a very slow learning rate.

Benchmark

To standardise the evaluation of classification results obtained with the database, a benchmark based on f1 scores is proposed. So, we have picked pretrained VGG-16 model with simple logistic regression as a benchmark and try to beat the benchmark with hyperparameter turning. We will also try Ensemble methods if the hyperparameter tuning does not improve the score.

A well-designed convolutional neural network should be able to beat the random choice baseline model easily considering even the simple logistic regression model clearly surpasses the initial benchmark. However, due to computational costs, it may not be possible to run the transfer learning model with VGG-16 architecture for sufficient number of epochs so that it may be able to converge.

III. Methodology

Data Preprocessing

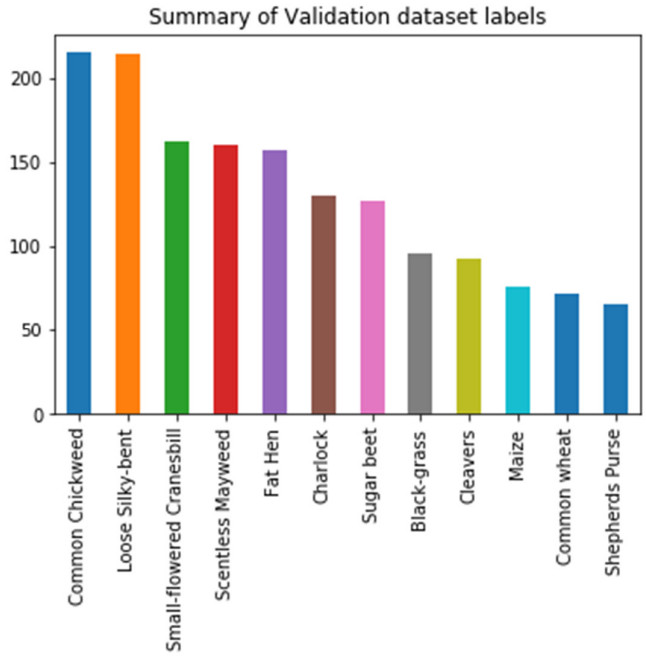
Preprocessing of data is carried out before model is built and training process is executed.

Following are the steps carried out during preprocessing.

1. Initially the images are divided into training and validation sets.
2. The images are normalised by dividing every pixel in every image by 255.
3. The images are resized to a square images i.e. 224 x 224 pixels for VGG-16 and 299 x 299 pixels for Xception pre-trained models.
4. All images were transferred using OpenCV and Image augmentation process got extracting image features.
5. Trained the model using pre-trained architectures (i.e. VGG-16, Xception) without fully connected layers.
6. Predict labels using supervised learning models (i.e. Logistic Regression, Random Forest) on Training and validation datasets.
7. Derive loss function value and accuracy values.
8. Finetune parameters and repeat step 4. To step 7. Until get optimized results.

As per using VGG-16 architecture for transfer learning, images are preprocessed as performed in the original VGGNet paper. Creators of the original VGGNet subtracted the mean of each channel (R,G,B) first so the data for each channel had a mean of 0. Furthermore, their processing software expected input in (B,G,R) order whereas python by default expects (R,G,B), so the images had to be converted from RGB -> BGR. In this dataset input images also come in different sizes and resolutions, so they were resized to 240 x 240 x 3 to reduce size. Dataset given by Kaggle does not have any validation set, so it was split into a training set and a validation set for evaluation. Out of 4750 images, 3182 images are in the training set and the remaining 1568 (0.33% of all classes) are

in the validation set. By using train_test_split methods in scikit-learn it randomly took 0.77% of each classes from the training set to the validation set while preserving the directory structure. It preserves the distribution of the classes as visualized below.



OpenCV and Keras ImageDataGenerators generate training data from the directories/numpy arrays in batches and processes them with their labels. Training data was also shuffled during training the model, while validation data was used to get the validation accuracy and validation loss during training. For image preprocessing, transformation and augmentation implemented with all below features and function.

Image Resizing: Before applying any technique in deep learning model first step is to resizing images. Most of the cases images gathered from Internet or real taken photographs will be of varying sizes. Due to presence of fully connected layers in most of the neural networks, the images being fed to network will be required of a fixed size (unless you are using Spatial Pyramid Pooling before passing to dense layers). Because of this, before the image augmentation happens, we need to do preprocess the images to the size which our network needs. With the fixed sized image, we get the benefits of processing them in batches. Image sizing process happened two times. One process is image size fixed with (244, 244, 3) for pretrained VGG-16 model, another size with (299, 299, 3) for Xception and InceptionV3 pretrained models.

Image Transformation: Image transformation has done by using OpenCV tool for all train and validation images.

Image Mask: By using Morphological Transformations we can perform that some simple operations based on the image's shape.

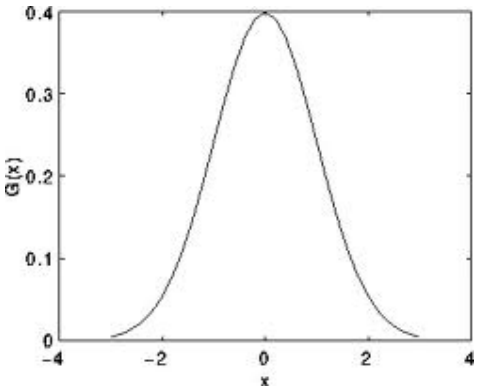
Here we have used pair called "opening" and "closing" pair. Closing is reverse of pening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black

points on the object. The goal with opening is to remove "false positives" so to speak. Sometimes, in the background, we get some pixels here and there of "noise." The idea of "closing" is to remove false negatives. Basically, this is where we have plant seeds detected shape, like seed leap, and yet we still have some black pixels within the object. Closing will attempt to clear that up.

Image Segment: This includes bitwise AND, OR, NOT and XOR operations. Here we have used bitwise AND operations. These features will be highly useful while extracting any part of the image, defining and working with non-rectangular ROI etc. Below we will see an example on how to change a particular plant seed of an image.

Image Sharpen: By using GaussianBlur and addWeighted functions, we can apply diverse linear filters to smooth images to get sharpen.

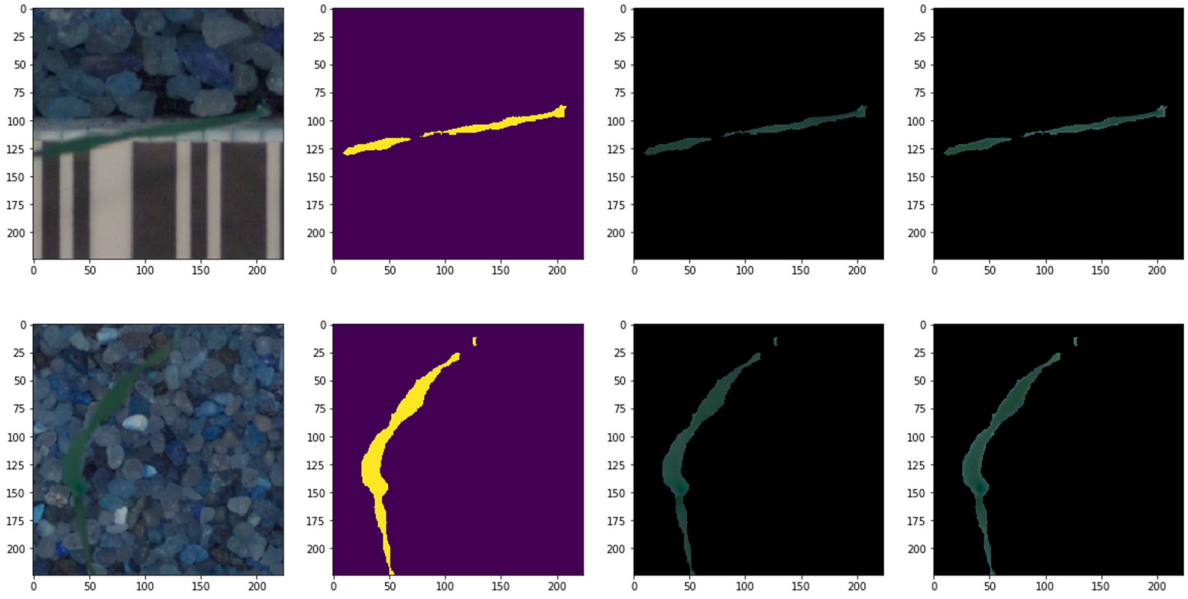
GaussianBlur: Here first we do an image blurs using a Gaussian filter function. The Gaussian filter is a low-pass filter that removes the high-frequency components are reduced. Probably the most useful filter (although not the fastest). Gaussian filtering is done by convolving each point in the input array with a *Gaussian kernel* and then summing them all to produce the output array. Just to make the picture clearer, remember how a 1D Gaussian kernel look like?



Assuming that an image is 1D, you can notice that the pixel located in the middle would have the biggest weight. The weight of its neighbors decreases as the spatial distance between them and the center pixel increases.

addWeighted: This function add two images which we have linear blend images from above. It calculates the weighted sum of two arrays. where image is a multi-dimensional index of array elements.

Bu doing all the above image transformation we are extracting more features and feeding to train model. Below showed sample images after image transformation.



Data augmentation:

Deep algorithms need large amount of training data to achieve good performance. To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc. This helps prevent overfitting and helps the model generalize better.

Image Data Generator:

An augmented image generator can be easily created using ImageDataGenerator API in Keras. ImageDataGenerator generates batches of image data with real-time data augmentation. The pre-trained VGG-1 deep neural network with augmented images are as follows below transformations:

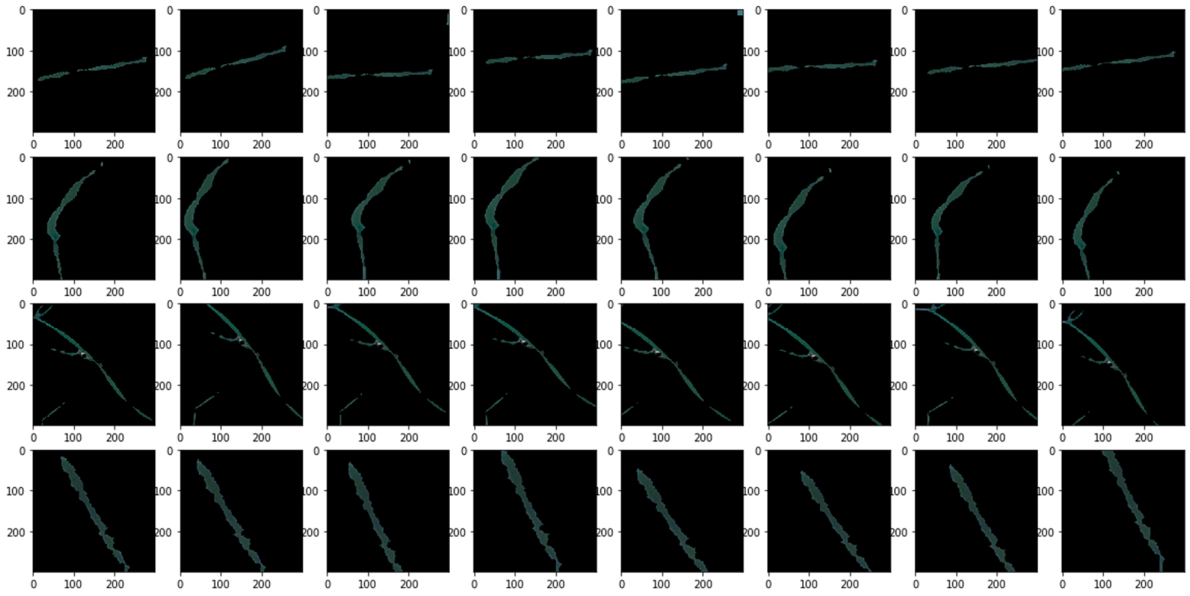
Random Rotation: Sometimes images in sample data may have varying and different rotations in the scene. We can train our model to better handle rotations of images by artificially and randomly rotating images from your dataset during training. In training and validation data sets, images have been minute angle rotated left and right up to a limit of 10 degrees. This transform might be of help when learning from photographs where the objects may have different orientations.

Random Shift: Objects in plant seeds images may not be centered in the frame. They may be off-center in a variety of different ways. We can train our model to expect and currently handle off-center objects by artificially creating shifted versions of our training data. Keras supports separate horizontal and vertical random shifting of training data by the width_shift_range and height_shift_range arguments. In our model we have shifted minute angle to each image by width_shift_range = 0.1, height_shift_range = 0.1. This can be useful on more complex problem cases.

Random Flip: Another augmentation to plant seeds image data that can improve performance on large and complex problems is to create random flips of images in our training data. Keras supports random flipping along both the vertical and horizontal axes using the vertical_flip and horizontal_flip arguments. This transformation may be useful for problems with photographs of objects in a scene that can have a varied orientation.

This data preparation and augmentation is performed just in time by Keras. This is efficient in terms of memory, only thing is we need to feed more required and the exact images to model during training.

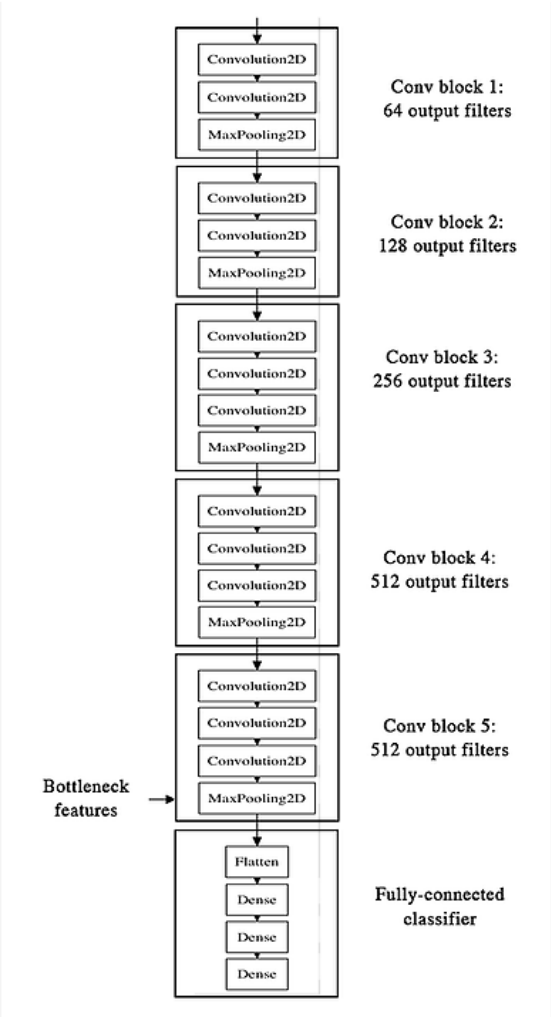
Below image is result of after Image Augmentation transformation.



By looking at above sample images, we can assure that now our model can capture edges and read more number of features related to each plant seeds and will make our model prediction accuracy.

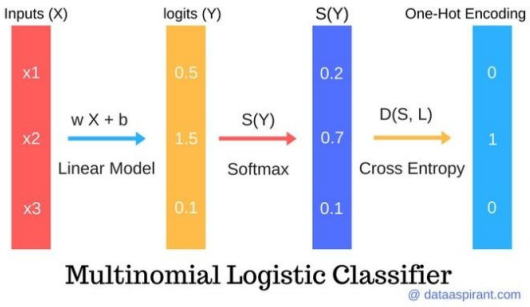
Implementation

After images were split into a training set and a validation set, the training data has been pretrained model by VGG-16 with Logistic Regression were implemented for comparison. Pre-processing operations such as image transformation process and subtracting the mean of each of the channels as mentioned previously was performed and VGG-16 architecture without the last fully connected layers was used to extract the convolutional features from the preprocessed images. VGG16 architecture diagram without the fully connected layer is given below:



On the extracted bottleneck features (CNN convolution + pooling operations), followed by a number of fully connected layers. Here is we are performing multiclass classification the output is softmax using logistic regression to predict plant seed labels. But unfortunately, it didn't have a good result on this bottleneck features which we obtained log loss is 0.80% and validation accuracy is 0.74% whereas train data accuracy is 0.77%.

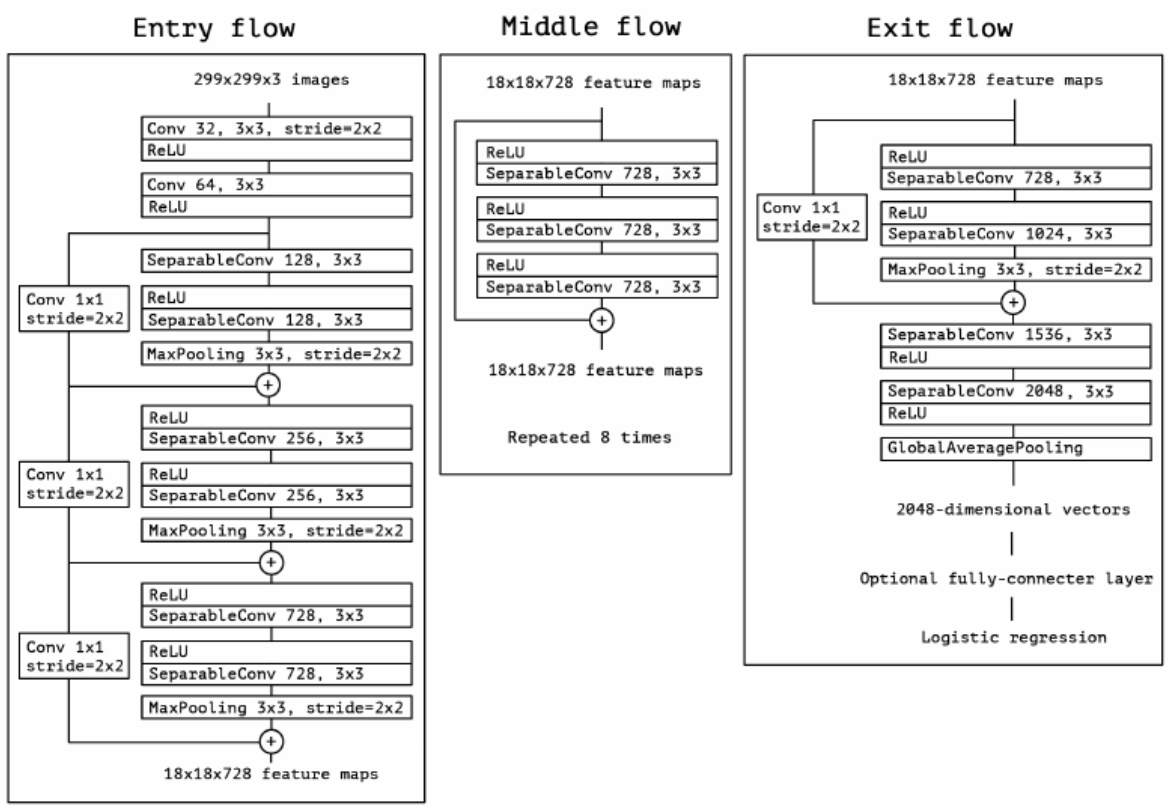
Here is how multinomial logistic classifier is derived target label by taking input from above extracted bottleneck featres.



I mentioned in the proposal that I'd be trying a logistic regression model on the CNN bottleneck extracted features, however later it seemed that'd result in too many weaker models and since the

aim of the project is to establish a transfer learning model, it's better to focus on that more. That is why for extracting more features image transformation and image augmentation pre-processed transformation were functional done.

For getting better results we need to play around multiple pretrained bottleneck architectures. By doing this following bottleneck feature is Xception. In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. “Xception” means “Extreme Inception”, as this new model uses depthwise separable convolutions, which are at one extreme of the spectrum described above VGG-16.



After pretrained data with Xception bottleneck features. This time model accuracy improve lot compare with previous results. This time we obtained logloss value is gone down to 0.28% and validation accuracy gone up to 0.90% whereas training data accuracy move to 0.99.9%. This is impressive improvement.

Refinement

By choosing best pretrained bottleneck model. We played training with Xception and InceptionV3 architectures. And for better classification model we have applied logistic regression and Random Forest models on to of pretrained Xception and InceptionV3 architectures. Even in the model with batch-normalization enabled during some epochs training accuracy was much higher than validation accuracy, often going near 99.9% accurate. Since the data set is small (only 4750 training images) it's definitely plausible our

model is memorizing the patterns. To overcome this problem, data transformation and data augmentation was used. Data transformation alters our training data by applying image mask, image segment, & image sharpen etc. At the same data augmentation alters our training batches by applying random rotations, cropping, flipping, shifting, shearing etc.

Unfortunately enough, the model with data augmentation is computationally expensive and takes around 1 hour per each pretrained model. It may be desirable to use a large batch size when training the pretrained and a batch size of 1 when making predictions in order to predict the next step in the sequence. In our model we have taken batch_size=32 sample. Specifically, the batch size used when fitting your model controls how many predictions we must make at a time. After refined all these features our log loss value decrease from 74% to 28% and validation accuracy improved from 77% to 90%.

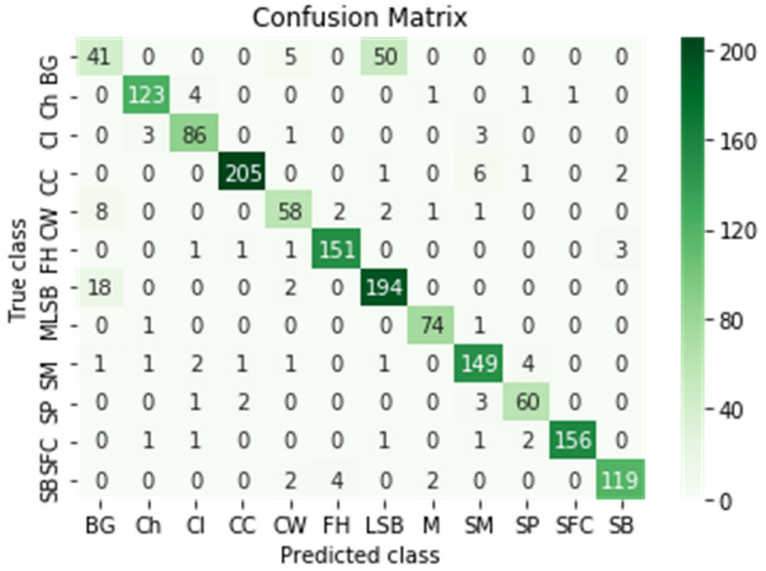
IV. Results

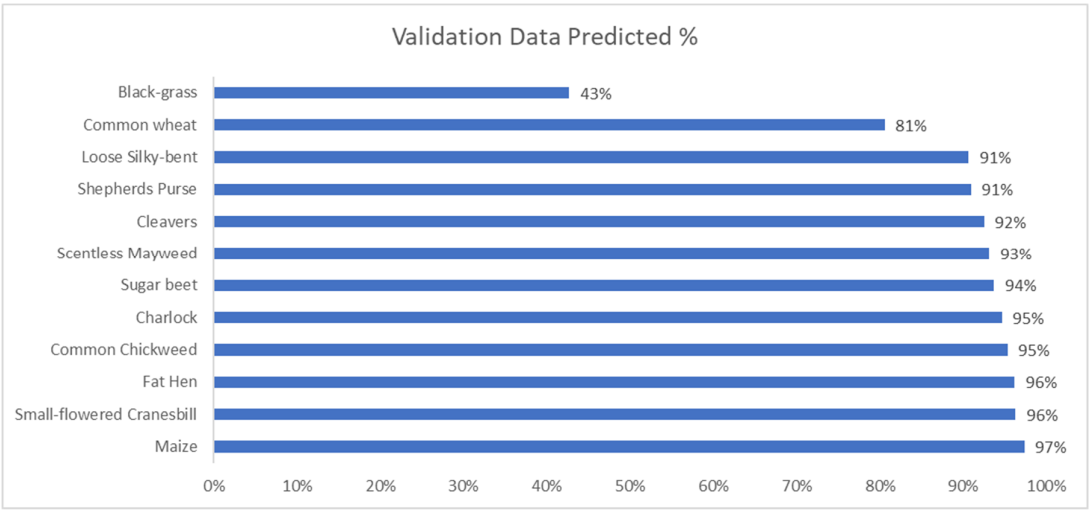
Model Evaluation and Validation

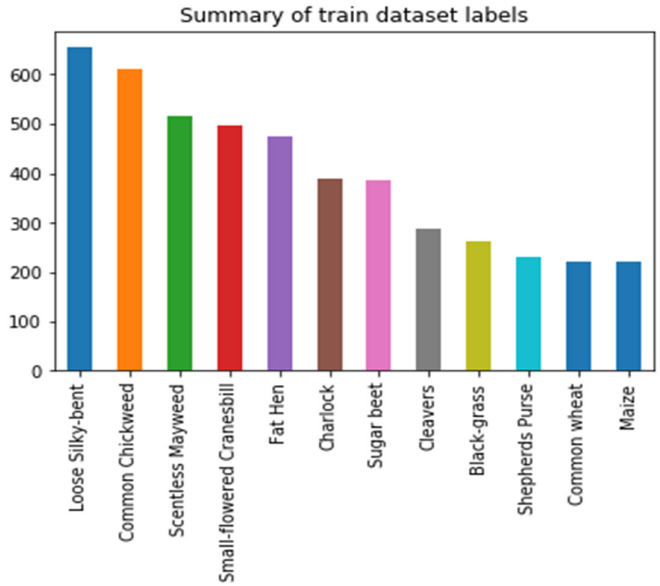
Final model has been chosen by Xception excluding the fully connected layers and then applied logistic regression to predict the final images label. This model beats the all combination (pretrained + classification model) of models including which we have chosen benchmark model VGG-1 with logistic regression. A table with all the experiments performed is given below along with their results.

Trained Model	**Log Loss**	**Accuracy** F1- Score (micro)
VGG + Logistic Regression	0.43	0.86
Xception + Logistic Regression	0.28	0.89
Xception + Random Forest	0.28	0.90
InceptionV3 + Logistic Regression	0.45	0.87
InceptionV3 + Random Forest	0.34	0.88

In the validation data out of 1568 images, 1405 images are classified accurately and 163 images are incorrect. The confusion matrix(non-normalized) plot of the predictions on the validation data is given below.

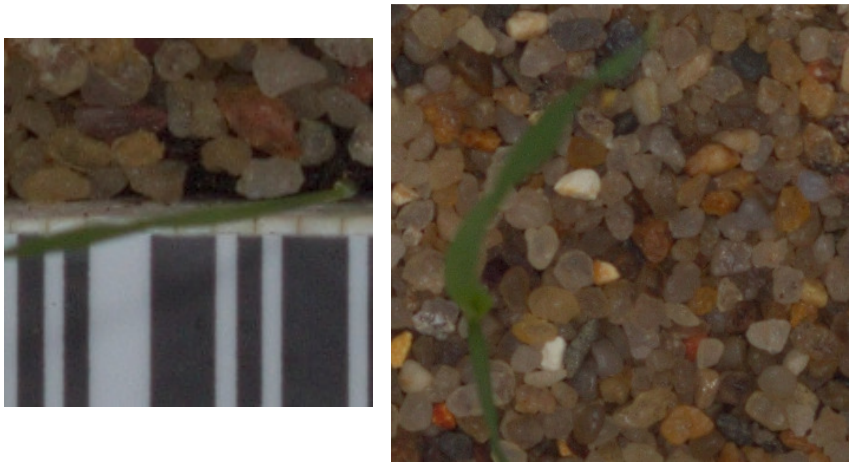

Confusion Matrix
True class
Predicted class
BG Ch Cl CC CW FH LSB M SM SP SFC SB
41 0 0 0 5 0 50 0 0 0 0 0
0 123 4 0 0 0 0 1 0 1 1 0
0 3 86 0 1 0 0 0 3 0 0 0
0 0 0 205 0 0 1 0 6 1 0 2
8 0 0 0 58 2 2 1 1 0 0 0
0 0 1 1 1 151 0 0 0 0 0 3
18 0 0 0 2 0 194 0 0 0 0 0
0 1 0 0 0 0 0 74 1 0 0 0
1 1 2 1 1 0 1 0 149 4 0 0
0 0 1 2 0 0 0 0 3 60 0 0
0 1 1 0 0 0 1 0 1 2 156 0
0 0 0 0 2 4 0 2 0 0 0 119
200
160
120
80
40
0


Validation Data Predicted %
Black-grass 43%
Common wheat 81%
Loose Silky-bent 91%
Shepherds Purse 91%
Cleavers 92%
Scentless Mayweed 93%
Sugar beet 94%
Charlock 95%
Common Chickweed 95%
Fat Hen 96%
Small-flowered Cranesbill 96%
Maize 97%
0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%


Summary of train dataset labels
600
500
400
300
200
100
0
Loose Silky-bent
Common Chickweed
Scentless Mayweed
Small-flowered Cranesbill
Fat Hen
Charlock
Sugar beet
Cleavers
Black-grass
Shepherds Purse
Common wheat
Maize

By observing above confusion matrix and validation prediction graph, this model is really good at predicting except “Black-grass” and “Common wheat” respectively, seemingly because the training data provided by Kaggle itself has less images for these two plant seeds compare to other classes. Interestingly, “Maize” class has very less samples compare to other class, but prediction accuracy is higher among others. That means accurate is not reflection of the volume of image in that “Maize”. The main challenge of identifying “Black-grass” and “Common wheat” seed class are more tough judgement being a human due to their appearance and image quality which shows below. Particularly “Black-grass” has very low prediction rate which is 43% only. This is very poor prediction rate in classification problem. I am sure if we do more image fine tuning by image transformation and augmentation process and feed more features to model these two classes prediction will be improve.

Few samples of “Black-grass”



Few sample of “Common Wheat”

Justification

There is room for improvement on the final results, the tuned final model significant enough improvement over the untuned model. There could be more ways we could improve the score, particularly by selecting by other pretrained model with other classification model and then performing the same exercise we did as describe above. But this will be out of scope of this report for now.

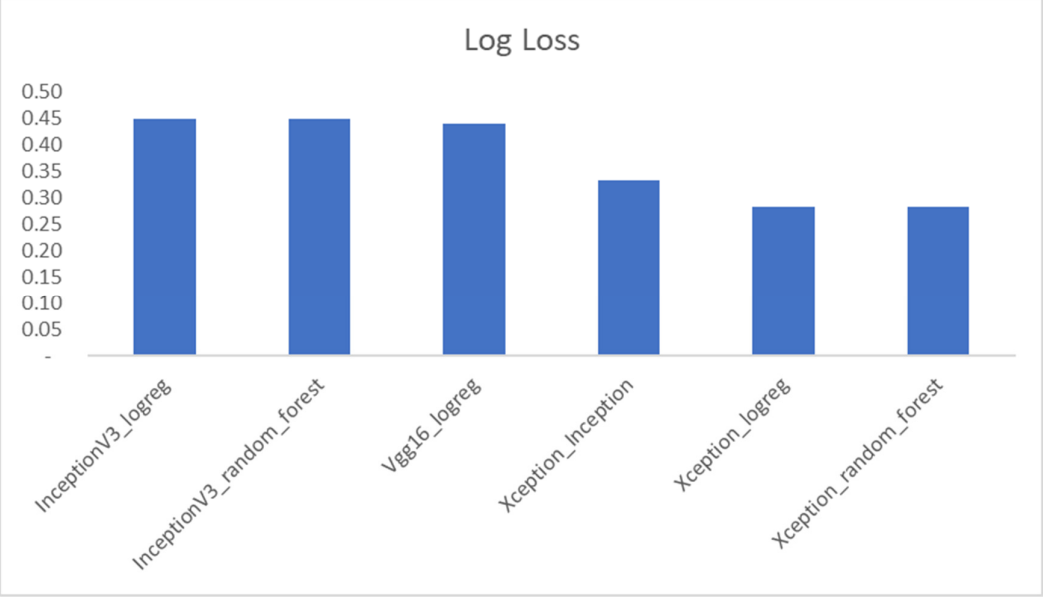
	Benchmark model **VGG + Logistic Regression**	**Final Model** **Xception + Logistic Regression**
F1-Score	0.86	0.90

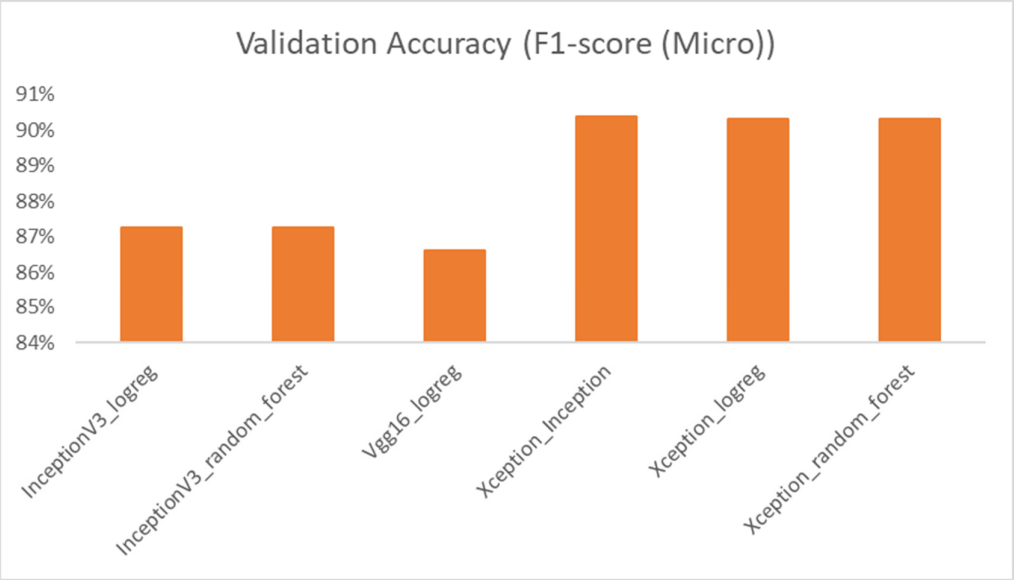
V. Conclusion

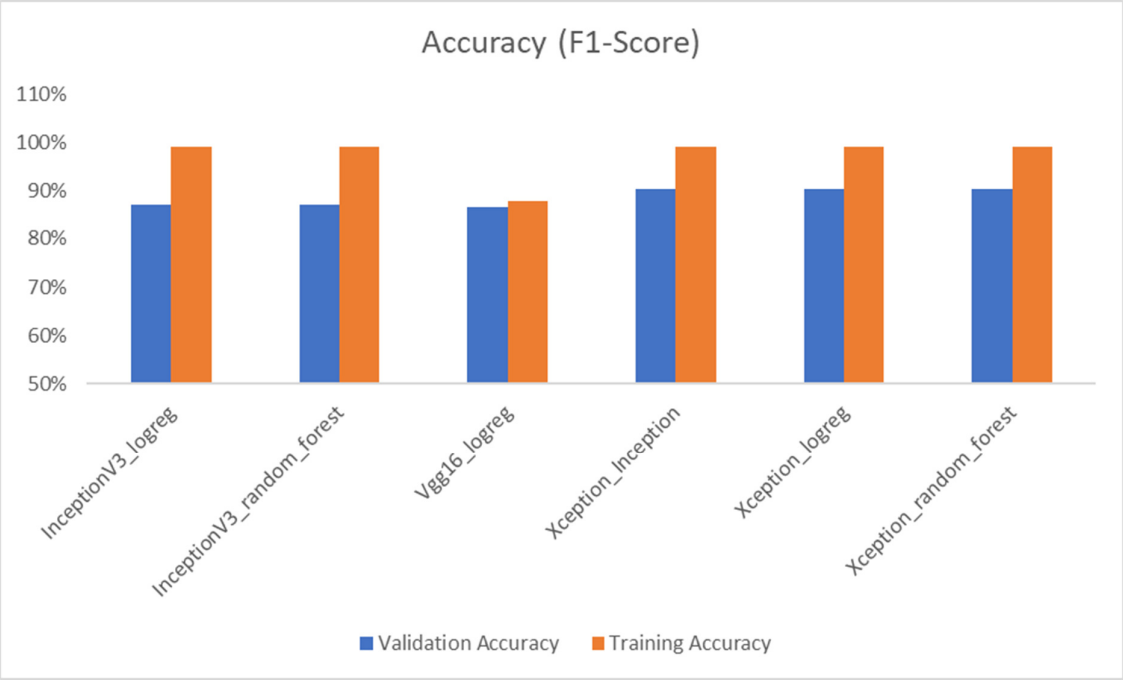
(approx. 1-2 pages)

Free-Form Visualization

For validation purpose all models accuracy and loss measures are recorded and here final model can be compared to the second best alternative. Here's the accuracy/loss graph of the model with batch normalization, and with data augmentation.







As we can see the VGG-16 validation accuracy is near 87% in the diagram and the loss is near 44%. Similarly the training accuracy is near 88%. At the same if we observed that that Xception with logistic regression validation accuracy is near 90% and the loss is 28%, and training accuracy is around 99%. We also see except VGG-16 with logistic regression all other combination models are showing accuracy 10% differentiation between training and validation data sets. Clearly this model is overfitting on the training data with data transformation and augmentation. We chosen Xception with logistic regression model is final model because of loss is less and accuracy is more compare to all other models.

Reflection

For reaching into this end to end solution, I've tried to progressively use more complex models to classify the images. The random forest and logistic regression with all three pretrained model, however they measured similarity against the average image of each class, whereas final one is Xception with logistic regression is final vote. I've even tried a baseline convolutional model as a good-practice because I wanted to see how the model performs with a conv model with a few number of layers only (it heavily underperforms unfortunately). To come to the point of using Data Augmentation, I had to extract the CNN features first and experiment with running different versions top layers on the CNN features. Only after applying batch normalization instead of the VGG-style fully connected model I saw significant improvement, and so I used it with the VGG architecture and applied data augmentation with it.

I believe still we can play around with image preprocessing parameters to provide more plant seeds feature to boost the performance of deep networks. The most difficult part for me was to get the experiments running on my local machine. Higher computational time results in lower number of experiments when it comes to neural networks, specially when I'm just figuring out what to do as it's my first experience with deep learning. However as I feel more confident in my skills in using deep learning now, I'll definitely try to seek more computational resources from now on.

Improvement

Following are the areas for improvement

1. To further improve the score, we need to consider using bigger size when re-sizing the image. Currently the algorithm was trained by using 224x224 and 299x299 re-sized images. An image size of 480x480 might be relevant as original image size is 640x480.

2. Currently VGG16, Xception and InceptionV3 architectures was investigated during Transfer learning. We also need to try using the below pre-trained models and verify the score.

- VGG-19 bottleneck features
- ResNet-50 bottleneck features
- MobileNet bottleneck features
- InceptionResNetV2 bottleneck features
- MobileNetV2
- DenseNet121
- DenseNet201
- NASNetLarge

3. For classification model we have used logistic Regression, Random Forest, but we can try other below classification models

- Linear Classifiers: Naive Bayes Classifier.
- Support Vector Machines.
- Decision Trees.
- Boosted Trees.
- Neural Networks.
- Nearest Neighbor.

3. Model Ensembles and use of advanced Image Segmentation algorithms such as R-CNN,Fast R-CNN,Faster R-CNN and Mask R-CNN can also be investigated.

Due to time and computational cost it was not possible for me to run more experiments using different known architectures other than VGG-16, Xception and Inception V-3 for this dataset. It's definitely possible that a different architecture would be more effective.

Limitations of platform

For Cloud Machine, the platform is hard to control and has less flexibility. Also the cost will be large if we only use enterprise cloud machine.

For Local Machine, it might be slow and short of memory because local computer do not always have a good CPU or RAM.

Reference

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, Karen Simonyan ∗ & Andrew Zisserman + Visual Geometry Group, Department of Engineering Science, University of Oxford {karen,az}@robots.ox.ac.uk

[1] Thomas Mosgaard Giselsson, Rasmus Nyholm Jørgensen, Peter Kryger Jensen, Mads Dyrmann: "A Public Image Database for Benchmark of Plant Seedling Classification Algorithms", 2017; [http://arxiv.org/abs/1711.05458 arXiv:1711.05458].

[2] Mads Dyrmann, Henrik Karstoft, Henrik Skov Midtiby: "Plant species classification using deep convolutional neural network", 2016; [https://www-sciencedirect-com.proxy.library.georgetown.edu/science/article/pii/S1537511016301465]