Machine Learning Engineer Nanodegree

Capstone Report: Plant Seedlings Classification in Kaggle

Yan Weili
March 18th, 2018

I. Definition

1.1 Problem Overview

Currently, farmers spray a standard herbicide mixture over the entire field to combat weeds [1]. Such a method is neither economical nor environment-friendly. The herbicides are not only expensive but also may pollute the rivers around. In addition, the effect of the standard mixture of herbicides varies on different weeds. Nowadays, several decision systems are able to reduce the herbicide expenditures by at least 40% by recommending the optimal herbicide dosages for a particular weed [1]. However, the category of the weed must be identified before using the decision system. Recently, a method using a fully convolutional neural network is presented to automatically detect the weeds in color images despite heavy leaf occlusion [2].

In order to automatically inspect the species of weeds (differentiate a weed from a crop seedling), 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 [3]. Kaggle are now hosting the Plant Seedlings Classification competition using this dataset to classify the different species based on their images. A set of images of plant seedlings at various stages of grown can downloaded at Kaggle [4].

1.2 Problem Statement

In this project, we will solely use the original training dataset (training.zip) at Kaggle [4] as the whole dataset for study.

We first need to download the dataset. In this dataset, each image has a filename that is its unique id. The dataset consists of 12 plant species, which is listed as:

Black-grass, Charlock, Cleavers, Common Chickweed, Common wheat, Fat Hen, Loose Silky-bent, Maize, Scentless Mayweed, Shepherds Purse, Small-flowered Cranesbill, Sugar beet.

In this project, we aim to train a convolutional neural network (CNN) with transfer learning to classify the plant's species from a photo. This is a multi-class classification problem. We will randomly choose 10% images in each species-named file as the testing dataset, and the resting 90% images in each species-named file will be used as the training and validation dataset.

1.3 Metrics

We apply the same evaluation metrics from Kaggle [5]. The prediction results are evaluated on *MeanFScore*, which is actually a micro-averaged F1-score at Kaggle. Given positive/negative rates for each class *k*, the resulting precision and recall scores are respectively calculated as

$$Precision_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FP_k}$$

$$Recall_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FN_k}$$

The micro-averaged F1-score *MeanFScore* is the harmonic mean of precision and recall, which is calculated as

$$MeanFScore = F1_{micro} = \frac{2Precision_{micro}Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

The micro-averaged F1-score sums up the true positives, false positives, and false negatives of all classes to compute the average score. As our problem is a multi-class classification one, the micro-averaged F1-score is suitable as the class may be imbalanced.

II. Analysis

2.1 Data Exploration

The data exploration part is done in Section "*1. Data Exploration and Visualization*" of the Jupyter notebook. Each image has a filename that is its unique id. Each image is stored in its corresponding species-named file. The dataset has 12 species-named files, and each file has more than 200 images. There are 4750 images in total (1.72GB) and there are no anomalies for all the images. Number of images, pixel mean, and pixel standard deviation for each plant species are listed in Table 1. The pixel values for all images are in the integers between 0 and 255.

Table 1. Statistics for each plant species

Class	Image count	Pixel mean	Pixel std
Black-grass	263	77.77932509	40.43612241
Charlock	390	73.6452993	37.92676923
Cleavers	287	67.90000533	29.91027576
Common Chickweed	611	71.1840074	33.09396395
Common wheat	221	76.41140241	37.32410053
Fat Hen	475	70.59756358	34.63926891
Loose Silky-bent	654	76.101788	38.78424718
Maize	221	77.35261712	39.9625301
Scentless Mayweed	516	70.53365124	31.62689063
Shepherds Purse	231	69.19659076	34.80213899
Small-flowered Cranesbill	496	69.46209855	33.86482312
Sugar beet	385	76.59535587	36.86744397

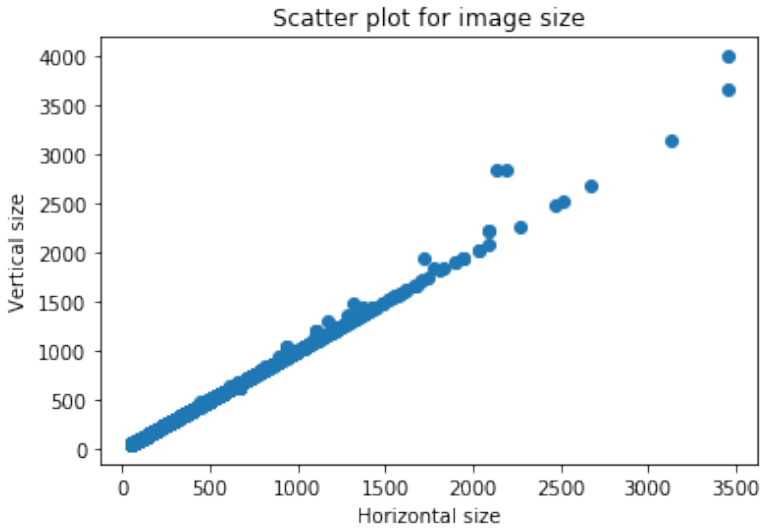

Figure 1. Scatter plot for image size.

Each image has different size and is colorful with three channels RGB. Scatter plot of the horizontal size verse vertical size for each image are shown in Fig. 1. Some samples for some seedling classes are shown in Figs 2-3. As can be seen, the image size is different even under the same class. In the data preprocessing part, we will reshape each image into the dimension of 224*224*3 for the VGG19 model [6].

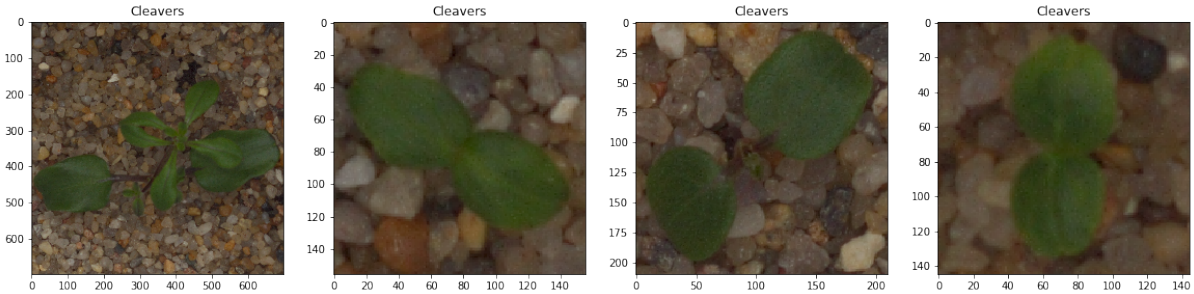

Figure 2. Some samples for the seedling class “Cleavers”.

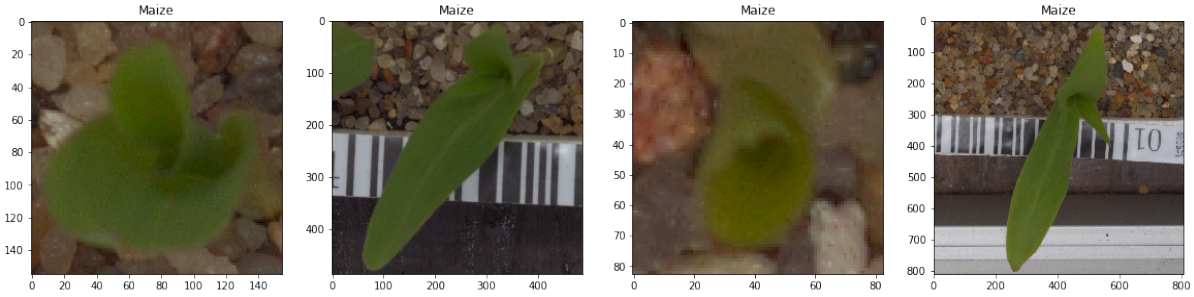

Figure 3. Some samples for the seedling class “Maize”.

2.2 Exploratory Visualization

This part is also done in Section *"1. Data Exploration and Visualization"* of the Jupyter notebook. Histograms of pixel intensity of three channels for two classes are respectively shown in Figs 4-5. As can be seen, the pixel distribution for different class is different. One may also use K-nearest neighbor method to classify different classes based on the histogram features.

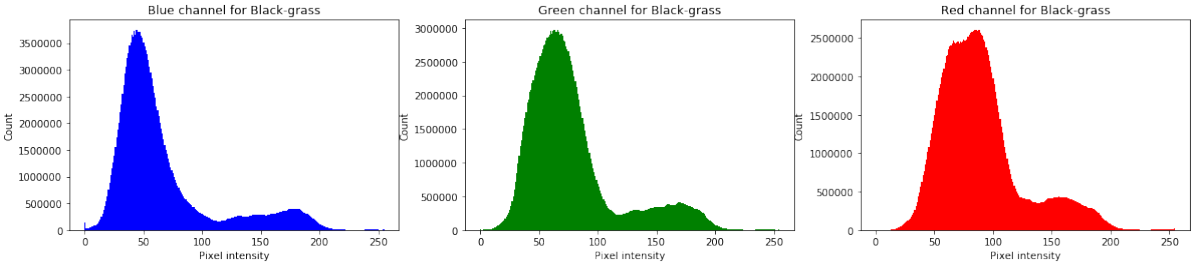

Figure 4. Histogram of pixel intensity of each channel for the seedling "Black-grass".

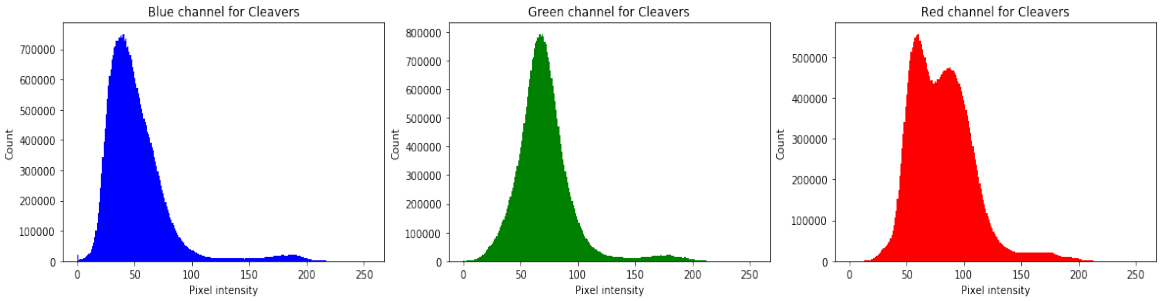

Figure 5. Histogram of pixel intensity of each channel for the seedling "Cleavers".

Next, the percentage of each class is shown in the Fig 6. As can been, each class is not quite imbalanced. Therefore, all the original images will be used in this project. In the data preprocessing part, we will also use the stratified sampling method to keep this class distribution.

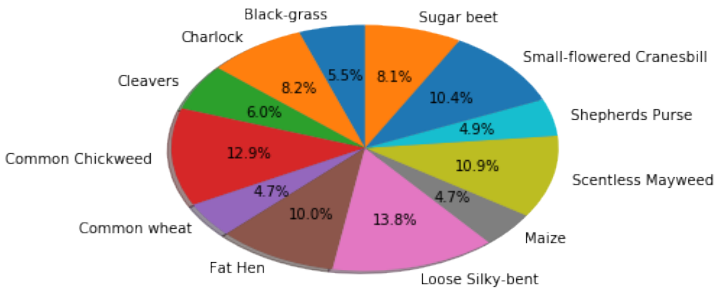

Figure 6. Pie chart for all classes.

2.3 Algorithms and Techniques

In this project, a classifier using convolutional neural network (CNN) is used as the CNN is the state-of-art method for the image classification problem. As the number of the images in this project are limited, we will not train the CNN from zero. Instead, transfer learning is used to extract the bottleneck features. Transfer learning reuses the learned weights from the pre-trained networks on a different dataset. The CNN model based on the bottleneck features using transfer learning will be trained to classify different plant seedling. In this project, the weights from VGG19 model will be applied to extract bottleneck features in images. VGG19 is developed by the Visual Geometry Group from university of Oxford [6]. Concepts such as convolution, pooling, activation and dropout used in the CNN model are briefly introduced as follows.

- Convolution: convolutional layers do a convolution operation on the input and pass the results to the next layer. The hyper-parameters for this layer includes filter number, filter size, stride, padding and so on. As compared to the fully connected neural networks, the convolution operation drastically decreases the number of weights.
- Pooling: pooling layers further reduce the outputs of a cluster. For instance, max pooling uses the maximum value from each of a cluster of neurons from the previous layer [7]. Average pooling uses the average value from each of a cluster of neurons from the previous layer.
- Activation: activation layers do a nonlinear operation on the output of the previous layer. For instance, the rectified linear unit (Relu) activation function is $f(x) = \max(0, x)$. The softmax activation function maps the previous outputs into probabilities, where the summation of the probabilities is 1 [8].
- Dropout: dropout refers to only activate the neuron with a certain rate. Such strategy can prevent the neural network from overfitting.
- Dense: dense layers refer to the conventional fully connected neurons that map the output of the previous convolution and pooling layers into the corresponding class. The softmax activation function is often applied in this layer.

As compared to support vector machine used in the following benchmark section, the CNN model takes the spatial relationship of image pixels. As such, the results using the CNN model are expected to be better. Besides, data augmentation will also be used to further prevent the neural network from overfitting.

2.4 Benchmark

2.4.1 Random Guess

The random guess is included as the first benchmark model. As there are 12 classes, the *MeanFScore* for random guess is 8.33%.

2.4.2 SVM Classification

The Support Vector Machine (SVM) classifier serves as another benchmark model. The SVM classifier is to be trained on the flattened array of the images.

This part is done in in Section *"3. Benchmark with SVM"* of the Jupyter notebook. The main steps are:

- Flatten the colorful image after the original image is reshaped into 224*224*3. Reshape of image is done in Section *"2. Data Preprocessing"* of the Jupyter notebook. Each image after flatten has a dimension of 150528.
- Use Principle Component Analysis (PCA) to reduce dimension of the image. The first 1000 principle components are used to represent the original data. These 1000 principle components explain about 90% of variation in the original data
- Use the SVM algorithm to build the model. One-vs-One is applied for multi-class classification.

The *MeanFScore* using SVM for the training and validation data is respectively 100% and 13.80%. The *MeanFScore* using SVM for the testing data is 13.68%.

It is worth noting that the training, validation, and testing dataset segment is explained in the following "Data Preprocessing" section.

III. Methodology

Data Preprocessing

This part is done in in Section *"2. Data Preprocessing"* of the Jupyter notebook. The main steps are:

- Randomly choose 10% images in each species-named file as the testing dataset, and the resting 90% images in each species-named file are used as the training and validation dataset (Of the 90% images, 80% for training and 20% validation). We use the stratified sampling method. After splitting, we have 3420 images for training, 855 images for validation, and 475 images for

testing. To reproduce the same results, we have saved the image directory. This part is done in "2.1 training/validation/testing files splitting of the notebook.

- Read the image and reshape each image into 224*224*3. This part is done in "2.2 obtain image input tensor, label, image ID" of the notebook.
- Use one hot encoding technique to obtain the corresponding image class label. This part is done in "2.3 one hot encoding" of the notebook.

Implementation

This part is done in in Section "*4. CNN Model with Transfer Learning"* of the Jupyter notebook. The main steps are:

- Extract the bottleneck feature using the VGG19 model. We extract the features from the layer "block5_pool" [9]. The shape of the extracted feature is 7*7*512. This part is done in "4.1 extract bottleneck feature using VGG19" of the notebook.
- Construct and compile model. Based on the extracted features, we construct the fully-connected neural networks (FCNN). For the FCNN, the extracted features are the input layer, followed by a global average pooling layer, and the output layer is a dense layer with 12 neurons and soft max activation. There are 6156 parameters in total to be trained. We use the categorical cross-entropy as the loss function, rmsprop as the optimizer, and accuracy as the metric for training the model. This part is done in "4.2 model construct and compile" of the notebook.
- Train and validate the FCNN model. We use the aforementioned training dataset to train the FCNN model. The FCNN model is trained for 20 epochs with batch size of 32. The best weight is saved with best accuracy score on the validation dataset during one epoch. This part is done in "4.3 model train and validation" of the notebook.

Extracting the bottleneck feature in this stage is quite time-consuming. It takes about two hours when I use Mac with CPU. Later, I use Ubuntu with GPU (6G memory) and it only takes about one minute to extract the features.

The *MeanFScore* using this FCNN model is 94.94% for the training dataset and 80.94% for the validation dataset.

Refinement

This part is done in in Section "4.5 results after refinement" of the Jupyter notebook. The main steps are:

- Augment the image data for training. For each image, rotate 90, 180, and 270 degrees to produce 3 images. For each image, crop upper left, upper right, center, lower left, and lower right with margin of 44 to produce 5 images. Therefore, we use the rotation and cropping technique to produce another 8 images for each image. After augmentation, images for training become 3420*9 = 30780.
- Obtain the bottleneck features of the augmented images.
- Obtain the corresponding one-hot-encoding label for the augmented images.
- Construct and compile the refined FCNN model (FCNN_RF). For the FCNN_RF, the extracted features of the augmented data are the input layer, followed by a global average pooling layer and a dense layer with 256 neurons and relu activation. Then a dropout layer with 0.2 rate and the output layer are added. There are 131328 parameters in total to be trained. We also use the categorical cross-entropy as the loss function, rmsprop as the optimizer, and accuracy as the metric for training the model.
- Train and validate the refined FCNN model (FCNN_RF). Similarly, we use the augmented training dataset to train the FCNN_RF model for 20 epochs with batch size of 32. The best weight is saved with best accuracy score on the same validation dataset during one epoch.

The *MeanFScore* using this FCNN_RF model is 92.53% for the augmented training dataset and 87.25% for the validation dataset. As mentioned earlier, the *MeanFScore* using this FCNN model is 94.94% for the training dataset and 80.94% for the same validation dataset.

IV. Results

Model Evaluation and Validation

This part is done respectively in "4.4 model evaluation on testing dataset" and "4.5 results after refinement" for the original FCNN model and the refined FCNN model (FCNN_RF). The main steps are the same, which is:

- Load the best weights with best accuracy score on the validation dataset.
- Predict and calculate the *MeanFScore* for the testing dataset using the trained model.

After computation, the *MeanFScore* on the testing dataset using the previous FCNN model is 79.58%, while the *MeanFScore* on the same testing dataset using the refined FCNN (FCNN_RF) is 86.95%.

To recap, the final refined FCNN model uses the extracted features of the augmented data using VGG19 as the input layer, followed by a global average pooling layer and a dense layer with 256 neurons and relu activation. Then a dropout layer with 0.2 rate and the output layer are added. For this model, categorical cross-entropy as the loss function, rmsprop as the optimizer, and accuracy as the metric for training the model.

It is reasonable to say that the results of the refined FCNN model can be trusted as the model is evaluated on the unseen testing dataset. To further verify the robustness of this model, we have performed 5-fold cross validation on the augmented data. The *MeanFScores* for each validation fold and the above same testing dataset are given in Table 2. From Table 2, it can be calculated that the mean of *MeanFScore* for the validation fold is 85.88% and is 86.02% for the testing dataset. The standard deviation of *MeanFScore* for the validation fold is 1.20% and is 1.29% for the testing dataset. Therefore, the prediction results of the refined FCNN model is quite stable.

Table 2 Results using 5-fold cross validation

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
MeanFScore for each fold	88.08%	84.59%	85.04%	85.88%	85.79%
MeanFScore for testing dataset	85.89%	88.0%	85.26%	84.21%	86.74%

Justification

As mentioned earlier, the *MeanFScore* for the testing dataset using the random guess benchmark model is 8.33%, and the *MeanFScore* for the testing dataset using the SVM benchmark model is 13.68%. As compared to the scores of these two benchmark models, the *MeanFScore* of 86.95% on the same testing dataset using the refined FCNN model has been significantly increased.

Given the *MeanFScore* of 86.95%, we can say that our proposed method can correctly classify the 12 class plant seedlings in the majority of cases. However, there is still a gap of about 14% to reach the complete correctness. We still need more efforts to improve the performance of the CNN model.

V. Conclusion

Free-Form Visualization

As mentioned in the above “Justification” section, the prediction results using the proposed method still needs to be improved. In this section, we want to examine the detailed prediction results for each class. Fig. 7 shows the confusion matrix for the testing dataset.

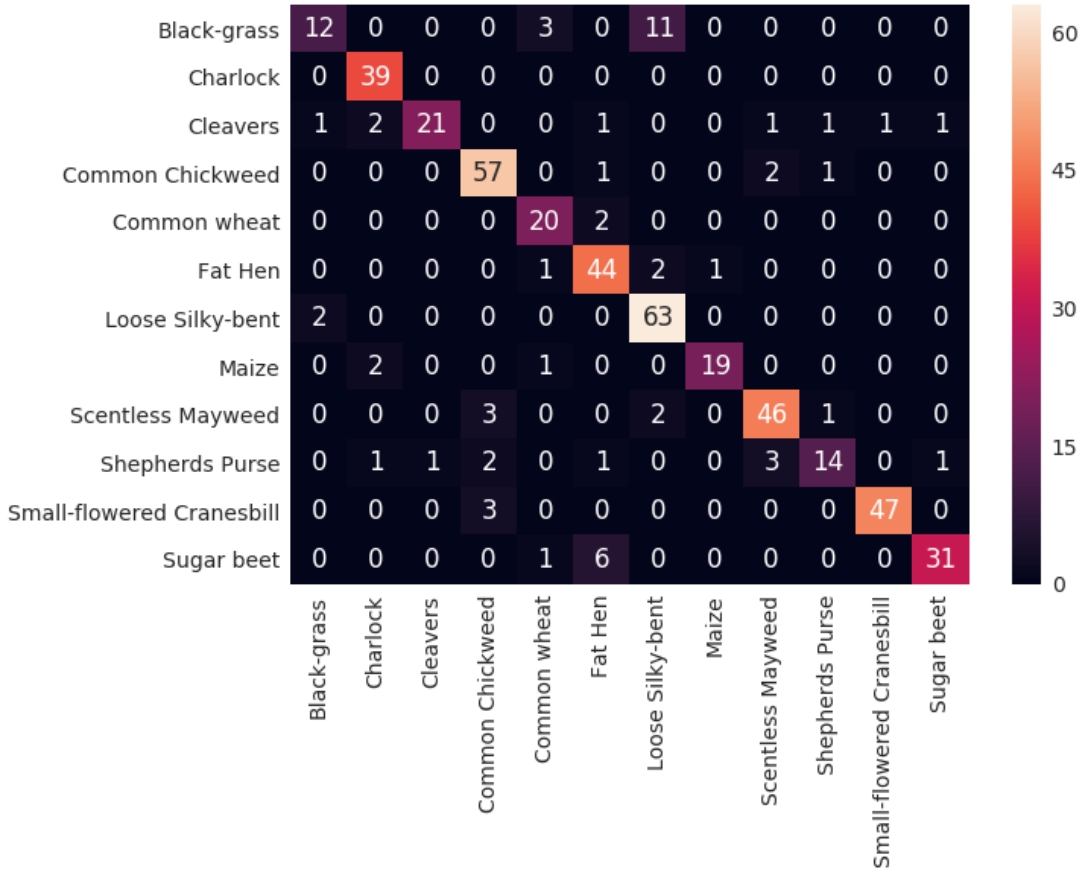

Figure 7 Confusion matrix for testing dataset

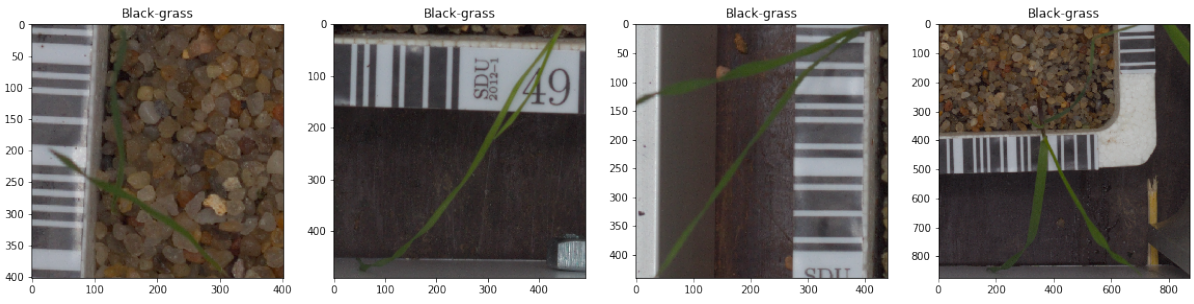

Figure 8. Some samples for the seedling class “Black-grass”.

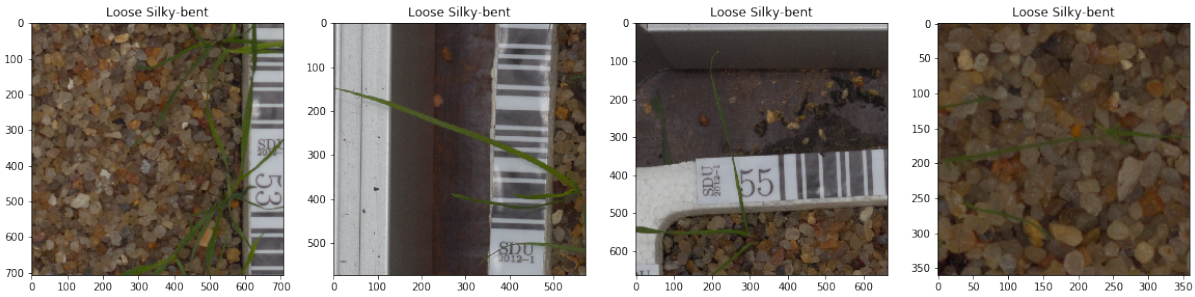

Figure 9. Some samples for the seedling class “Loose Silky-bent”.

As can be seen from Fig. 7, almost half of “Black-grass” has been mistakenly classified as “Loose Silky-bent”, and also some “Loose Silky-bent” has been mistakenly classified as “Black-grass”. We then show some samples for these two classes in Figs 8 and 9. It is not surprising that these two classes are very similar with each other. Moreover, the seedlings only take a little part of the whole image.

Therefore, in order to improve the performance of our model, we should need to do more data preprocessing on the original images. Simply reshaping may not be adequate.

Reflection

The process used for this project can be summarized using the following steps:

- Initialize the machine learning problem (whether it is a supervised, unsupervised or reinforcement learning one) and find the public dataset.
- Preprocess the dataset and define the suitable metrics.
- A benchmark model is designed for future comparison.
- Obtain the training, validation, and testing dataset based on the preprocessed dataset.
- Construct and compile the planned machine learning model. Try different structure and hyper-parameters.
- Train and evaluate the model several times until we obtain the best model for validation dataset.
- Evaluate the trained model on the testing dataset.

In this project, the most difficult part for me is to install many open source software packages. I particularly felt frustrated when I was installing the OpenCV package. Many version problem and missing files. I have to google many times to fix each problem. During that period, I realize that there is no free lunch on the earth. When the package is free, you have to solve the problem by your own.

I find interesting for this project is that I really felt the power of the transfer learning. We can easily reuse the weights of the pre-trained model of image classification to obtain a good classifier.

Improvement

In my opinion, the performance of the proposed method may be improved in the following aspects:

- Try different pre-trained models to obtain the bottleneck features. In this project, we only try VGG19. One can also try Xception, RESNET50, InceptionV3, or VGG16.
- Use the weights of the pre-trained model as the initial weight and train. In this project, we fix the weights of the pre-trained model. However, I am not sure whether retained the weights will improve the performance as the images are limited.
- More data preprocessing for the original images. We may further augment the training dataset and try other different techniques. I think this is the most promising aspect to improve the performance.

Reference

[1] https://vision.eng.au.dk/roboweedmaps/

[2] Mads Dyrmann, Rasmus Nyholm Jørgensen, Henrik Skov Midtiby. (2017,7). Detection of Weed Locations in Leaf-occluded Cereal Crops using a Fully-Convolutional Neural Network. Advances in Animal Biosciences Volume 8, Issue 2 (Papers presented at the 11th European Conference on Precision Agriculture (ECPA 2017).

[3] https://www.kaggle.com/c/plant-seedlings-classification#description

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

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

[6] K. Simonyan, A. Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv technical report, 2014.

[7] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). "Multi-column deep neural networks for image classification". 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE): 3642–3649. arXiv:1202.2745v1 Freely accessible. doi:10.1109/CVPR.2012.6248110. ISBN 978-1-4673-1226-4. OCLC 812295155. Retrieved 2013-12-09

[8] https://en.wikipedia.org/wiki/Softmax_function

[9] https://github.com/fchollet/deep-learning-models