Plant Seedling Classification with customized loss function and DCGAN

Xiao Yu*, Yi Ding*, Zhuoran Wu*, Joshuah Touyz
Computer Science Department
Georgetown University
Washington, D.C.
{yx151, yd137, zw118, jt1155}@georgetown.edu
*These authors contributed equally to this work.

Abstract

Plant Seedling Classification is a Kaggle competition with the goal to classify plant seedlings images into corresponding species. This project is a typical supervised, multi-class classification task. Convolutional Neural Networks (CNN) models perform really well on image classification task and therefore are adapted in this project. In this paper, both pre-trained models and customized CNN models were tested. Deep Convolutional Generative Adversarial Networks (DCGAN) was used to generate shaper images to alleviate the influence of data imbalance, adaptive synthetic sampling and synthetic minority oversampling were employed. Furthermore, a customized loss function was designed to classify two similar classes. Experiment results show that customized CNN models with the combination of data augmentation, data cleaning and model ensemble techniques could achieve nearly state-of-art classification results. Code is available at https://github.com/WuZhuoran/Plant_Seedlings_Classification

Keywords- Plant Seedling Classification; Convolutional Neural Networks; DCGAN; Hierarchical Loss Function; Image Classification;

I. INTRODUCTION

For several decades, researchers have been worked on systems aimed at performing site-specific weed control. Although some systems are commercially available, a true commercial breakthrough of such systems is still yet to come.[1] With the goal to contribute to such system, we participated in a Kaggle competition, Plant Seedling Classification, to design CNN models to classify 12 plants species. The best pre-trained model VGG19 with first five layers frozen was served as the baseline. We then designed a 9 layers CNN models and achieved a better result.

In this paper, the main contributions are 1) we modify the loss function according to Hierarchical Loss Function instead of traditional cross entropy. 2) we apply DCGAN in data augmentation part in order to get pictures with better quality. Furthermore, we also apply data cleaning, data balanced and model ensemble methods.

II. RELATED WORK

Transfer Learning has been proven useful for image classification task, especially for small dataset[11]. Since pre-trained models are trained on ImageNet, they are sensitive to certain features such as edges and suitable for classifying RGB images such as the Plant Seedlings Dataset. Additionally, Hinterstoisser indicated a simple trick to get a performance boost is to freeze the feature extractor and train only the remaining layers[12]. Applying pre-trained models could be a good start.

Hierarchical Softmax is widely used in the NLP. In[13], we find out that Hierarchical Softmax can also be used for classification problems. And we design a loss function base on the win matrix[13] which could satisfy our idea that the closer the prediction in the tree, the smaller the loss value.

The size of more than half of images in our training set are smaller than 256*256 pixels. Directly resize these images will produce some very blurry images. Then we are applying DCGAN(Deep Convolutional Generative Adversarial Networks)[14] for data augmentation. Instead of generating images with label, we are trying to upsample images base on DCGAN.

III. DATA ANALYSIS

A. Data Introduction & Evaluation Metric

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has released a dataset containing RGB images of approximately 960 unique plants belonging to 12 species at several growth stages[2] .

This is the only data source that will be used in our project and is publicly available online[2].. Following are sample images for each species: Figure 3. The size of each species is shown in the pie chart: Figure 1.

Our project is a supervised learning task. The training set of images is organized by plant species in 12 seperate folders. Each image has a unique id that can be easily linked

to its plant species. The testing set of images is a mix of 12 plant species without labels.

We selected categorical cross entropy as our loss function since it is preferred for mutually-exclusive multi-class classification task (where each example belongs to a single class) compared to other metrics.

The micro-averaged F1 score was the major evaluation matrix in our project. All results were submitted to Kaggle competition for evaluation. We also used confusion matrix to visualize some prediction results.

B. Data Analysis

First, there are a few limitations of this dataset. First, the size of training set is relatively small. On average, there are about 400 images in each class. Second, the dataset is imbalanced. The largest class Loose Silky-bent has 654 images, but the smallest class Common Wheat only has 221 images. In this case, models tend to predict larger class. Third, since images with single plant seedling are cropped from a large image with multiple plant seedlings, image sizes vary a lot.

Before we build Deep Learning Model, we perform data analysis on training set. We first discover that the dataset is very imbalanced. According to Figure 1, the 12 species of seedling are widely distributed so that we need to balanced data in pre processing part.

Also, we found that not all the image are square, a few(about 0.1 percent) of pictures are rectangle so that we need to cut the image to square before directly resizing.

Furthermore, we apply PCA[9] and t-SNE[10] on the dataset. We first use PCA to lower our dimension to 180, then use t-SNE to 2 dimensions. According to the result scatter graph. We could see that different kinds of leaves are divided not very clearly. It shows that we could not use traditional methods to solve this problem perfectly. This is why we apply Deep Learning on this task.

IV. Methodology

C. Data Preprocessing

The Dataset is imbalanced and noisy. We need to deal with the leaves themselves without noisy items such as bar code, background soil and pebbles. So we need to remove background and deal with imbalance data.

Removing background is easy this time because all the seedlings in the dataset are all green in color, while at the same time other background noise are black/white or brown. Therefore we apply CIELUV Color space[3] as our distance metrics. Then we apply a median filter to remove artifacts. Examples of preprocessing show as Figure 1.

Considering that we are facing a imbalanced dataset, we apply Adaptive synthetic sampling approach for imbalanced learning (ADASYN) and Synthetic minority Over-sampling Technique (SMOTE). Also, we also apply the naive Random Up-Sampling as comparison. ADASYN generates synthetic data for classes with less samples in a way that datasets that are more difficult to learn are generated more compared to samples that are easier to learn[4]. SMOTE involves over sampling the minority class and under-sampling of the majority class to get the best results[5].

The visualization for different balance methods is shown in Figure 2 Left.

D. Convolutional Neural Networks (CNN)

We selected multiple pre-trained models from Keras. They are VGG19, ResNet50, Xception and Inception ResNet V2. Compared to other pre-trained models, these four models performed well as feature selectors with modified connected to fully connected layers. Stacking different pre-trained models together could combine the strength of both models and performed better than single pre-trained models. We also tried to freeze first ¼ to ⅛ layers to keep basic features and trained the rest of layers. Overall, freezing layers yielded better results/F1 sores.

In addition to make use of pre-trained models, we also built our own CNN model. Our model had 7 layers of convolutional layers and 2 layers of dense layers. Each convolutional layer was followed by a batch normalization layer and a LeakyReLU component. Each dense layer was followed by a dropout layer, a batch Normalization layer and activation functions of Tanh and Softmax. Apart from the Hierarchical Loss Function part, all the loss function was categorical cross entropy. Detailed configurations are shown at Table IV.

E. Hierarchical loss function

From the classification results, we noticed that two classes, Black-grass and Loose Silky-bent, were often misclassified. These two species are visually similar and posed a challenge to models. Therefore, we designed a customized loss function to improve the result. We define a Huffman tree with the weight of root is 0.5, the weight of children is 0.5 times than its parent and the weight of leaves is same as its parent. Then define the loss function:

$$Loss = -log(W')$$
$$W' = 2 * (W - 0.5)$$
$$W = S^T * C$$

Where S denote the softmax vector and C denote the vector of weights we defined.

F. Data Augmentations & Deep Convolutional Generative Adversarial Networks

We select images with a size larger than 256x256 pixels in the original training data, and reduced them to 256x256 pixels as the target image generated by CNN. At the same time, the original image is randomly reduced to size in the [50,128] pixels, and then enlarged to 128x128 as the input of the CNN. And base on DCGAN[14], we design a generator

and a discriminator. For generator we apply a decoder and a encoder. For discriminator we choose a CNN classifier to classify the real image and generated image.

G. Model Ensembling

Snapshot Ensemble Method is used for Model Ensemble. Snapshot Ensemble achieves the goal of ensembling by training a single neural network, and making it converge to several local minima along its optimization path and saving the model parameters[6]. The comparison of single model SGD schedule and Snapshot Ensemble shows as Figure 2.

V. Result

Considering that there is a Kaggle Competition for this task. We have a training set with 4750 images and a test set 794 images. We split 80% of dataset as training and 20% as validation set. Our experiments part is divided into 3 parts: Pretrained Model as benchmark, Simple CNN Model with customized Loss Function and Customized CNN Models with different pre and post processing. Then we use our model on our test data. Then we submit result file to Kaggle for the final F1 score.

H. Pretrained Model and Benchmark

Simply modified fully connected layers to predict 12 classes. Results shows at Table I. Best performance was achieved by Xception. Freeze about first ⅛ to ¼ layers for each models. Best performance was achieved by VGG19 with the first 5 layers frozen.

In Table II, select well performed pre-trained models and combine both, then add a convolutional layer, a batch normalization layer, a max pooling layer, and a dropout layer. The combination of VGG19 and Xception achieved good result.

TABLE I. Results of cnn

Models	**VGG19**	**ResNet 50**	**Xception**	**Inception ResNet V2**
Fine-tuning	0.64735	0.41939	0.69017	0.51259
With Frozen Layer	**0.79345**	0.54219	0.09571	0.56045
# of weights layers	19	50	129	777
Freezed # of weights layers	5	10	30	100

TABLE II. Results of combined model

Models	**VGG19+ Xception**	**Xception+ Inception ResNet V2**
F1 score	0.73173	0.55667

Overall, without DCGAN and median filter, VGG19 with the first 5 layers frozen was the best pre-trained model with 0.79345 F1 score. We selected it as our baseline model.

I. Hierarchical Loss Function.

We only define a fix tree structure for this training set and a fix multilayer CNN network for this loss function. Besides, since it quite hard to converge we apply some tricks such as load weights from other loss function. The result of our loss function is bad. The F1-score of our loss function is around 60% which is much lower than categorical cross entropy. But we also find that in specific cases (few short learning), our customized loss function had better performance than than categorical cross entropy.

J. Customized CNN

We compare results with different methods and the state of art performance on Kaggle Leaderboard. In the result table, 6 and 9 layers represents for 2 CNN models we used. The result show as Table III.

TABLE III. Results of cnn

Result Detail		
Method Used	***Parameters***	***F1***
VGG19	20,270,264	0.80
6 Layers	3,320,396	0.93306
6 Layers + Background Remove	3,320,396	0.95843
9 Layers + Fine Tuning	1,775,100	0.97103
9 Layers + Background Remove	1,775,100	0.97507
9 Layers + ADASYN	1,775,100	0.97799
9 Layers + Random Up Sample	1,775,100	0.97744
9 Layers + SMOTE	1,775,100	0.97832
9 Layers + ADASYN + Data Augmentation	1,775,100	0.97902
9 Layers + ADASYN + Data Augmentation + Snapshot Ensemble	1,775,100	**0.98740**
CNN DenseNet 121 + Data Augmentation + Background Remove	8,062,504	0.94710
CNN DenseNet 121 + Data Augmentation + GAN + Background Remove	8,062,504	0.97984
State of Art on Kaggle[a]	54,521,176	0.99496

a. Best Result until the end of competition.

VI. Result Analysis

In general, pre-trained models with first few layers frozen would yield better results than using entire models as

feature selectors because frozen layers were sensitive to features after trained on other dataset. However, it was difficult to choose the number of layers to freeze. Without DCGAN and median filter, VGG19 with first five layers frozen achieved best performance among all pretrained models. However, pre-trained models were not the best choice for the Plant Seedlings dataset. This model only served as a baseline model in our study.

For DCGAN, We try two different ways to generate the target image. The first one is repeatedly generate two times larger image till it larger than 256*256. The other way is use opencv resize to 128*128 and generate two time larger image. But we find both of them are not good enough. Then we try to train the model to generate 4 times larger image shown as Figure 4.And our results showed that the F1-score is 2% higher by using training set generated by DCGAN.

For our own CNN model, our results show that, after fine-tuning parameters, all the results are better than single pretrained model. Particularly, if we focus on our 9 layers CNN model, our best result is 0.98740 in F1 score. Compared with State of Art solution[8], we found that we only use a model with 7 times smaller than their model, which will reduce the training and testing time significantly. It shows that our model has a better ability to be built as a real-time system.

Furthermore, before fine-tuning, the background removal will help increase the model with 2 percent. However, after fine-tuning, the increase value is not big enough. It implies that background is not a key value for our whole mining task but remove noise is always good for task.

Also, in data balance process, we focus on comparison among Random Oversampling, SMOTE and ADASYN. We found that Naive oversampling is not quite useful. Both SMOTE and ADASYN made improvements in F1 score. And we decided to use ADASYN to do the next step experiment.

Then we apply the Model Ensemble. After snapshot ensemble, we found that the result increase around 0.01 and up to 0.987, which is a very good result.

VII. Conclusion

This work proposed plant seedling classification methods based on CNN. Based on data analysis results, we also applied several different pre and post possessing methods such as data augmentation and ensemble method. Furthermore, we proved and applied customized Loss function for better learning the model. And we apply DCGAN in data augmentation part. The final results shows that CNN is a great tool for image classification and we could reach 0.98-0.99 F1 score.

Acknowledgment

This research was instructed by Joshuah Touyz, Data Science Manager at Capital One and Adjunct professor at Georgetown University.

References

[1] Thomas Mosgaard Giselsson and (2017). A Public Image Database for Benchmark of Plant Seedling Classification. CoRR, abs/1711.05458.

[2] Giselsson, Thomas Mosgaard, Dyrmann, Mads, Jorgensen, Rasmus Nyholm, Jensen, Peter Kryger & Midtiby, Henrik Skov (2017). A Public Image Database for Benchmark of Plant Seedling Classification Algorithms.

[3] J. Schwiegerling, Field Guide to Visual and Ophthalmic Optics, SPIE Press, Bellingham, WA (2004).

[4] Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, 2008, pp. 1322-1328. doi: 10.1109/IJCNN.2008.4633969

[5] Chawla, N. V. et al. "SMOTE: Synthetic Minority Over-Sampling Technique." Journal of Artificial Intelligence Research 16 (2002): 321–357. Crossref. Web.

[6] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., & Weinberger, K. Q. (2017). Snapshot ensembles: Train 1, get M for free. arXiv preprint arXiv:1704.00109.

[7] Guillaume Lemaitre, Fernando Nogueira & Christos K. Aridas (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research, 18, 1-5.

[8] Kumar Shridhar., Kaggle #1 Winning Approach for Image Classification Challenge. Web Blog Post, Neural Space, June 20, 2018.

[9] Karl Pearson, 1901 K. Pearson, "On lines and planes of closest fit to systems of points in space", The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, Sixth Series, 2, pp. 559-572 (1901) (1857-1936)

[10] L.J.P. van der Maaten and G.E. Hinton. Visualizing Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

[11] Weiss, Karl, et al. "A Survey of Transfer Learning." *Journal of Big Data*, vol. 3, no. 1, 2016.

[12] Hinterstoisser, Stefan, et al. "On Pre-Trained Image Features and Synthetic Images for Deep Learning." 2017.

[13] Wu, Cinna, Mark Tygert, and Yann LeCun. "Hierarchical loss for classification." arXiv preprint arXiv:1709.01062 (2017).
5 Radford A, Metz L, Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks[J]. arXiv preprint arXiv:1511.06434, 2015.

[14] Gao F, Yang Y, Wang J, et al. A deep convolutional generative adversarial networks (DCGANs)-based semi-supervised method for object recognition in synthetic aperture radar (SAR) images[J]. Remote Sensing, 2018, 10(6).

Appendix

TABLE IV. CUSTOMIZED CNN MODEL CONFIGURATION

CNN Model Configuration	↓	↓
Input Image (51 × 51 RGB image)	Conv2d_4	Batch_normalization_7
Conv2d_1	Batch_normalization_4	LeakyReLU_7
Batch_normalization_1	LeakyReLU_4	**Max_Pooling2d_3**
LeakyReLU_1	**Max_Pooling2d_2**	Flatten_1
Conv2d_2	Conv2d_5	Dropout_1
Batch_normalization_2	Batch_normalization_5	Dense_1
LeakyReLU_2	LeakyReLU_5	Batch_normalization_8
Max_Pooling2d_1	Conv2d_6	Activation_1
Conv2d_3	Batch_normalization_6	Dropout_2
Batch_normalization_3	LeakyReLU_6	Dense_2
LeakyReLU_3	Conv2d_7	Batch_normalization_9
↗	↗	Activation_2

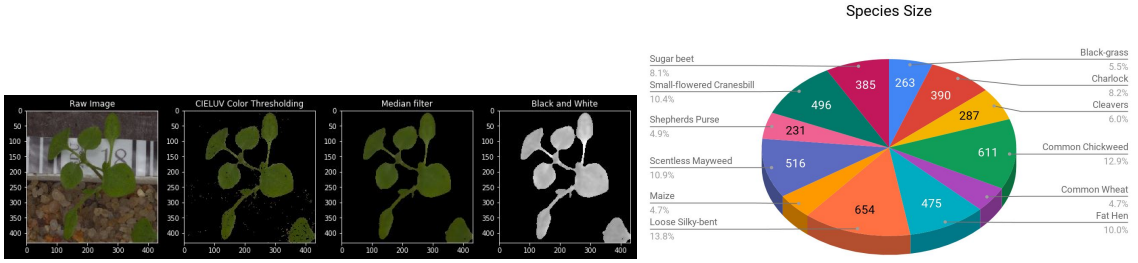

Figure 1. **Left:** Example of the Background Removal Process for image 495d1a520.png. **Right:** Species Size Pie Chart

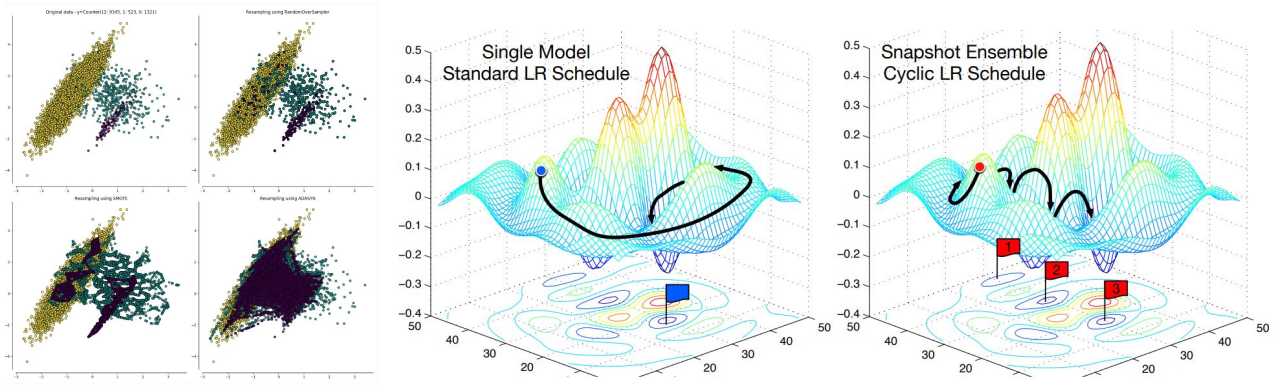

Figure 2. **Left:** Comparison for Random Up-Samping, SMOTE and ADASYN[7]. **Medium:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima[6]

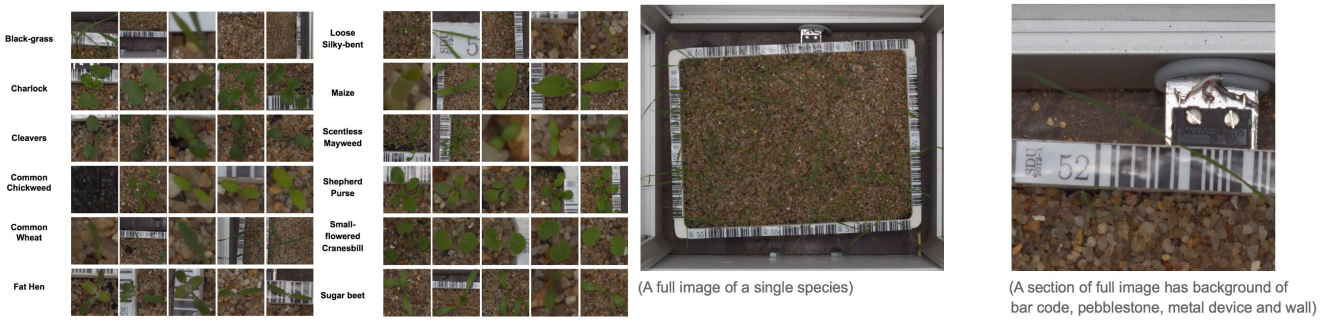

Figure 3. **Left**: Images of 12 Plant Species; **Right**: Noisy Background

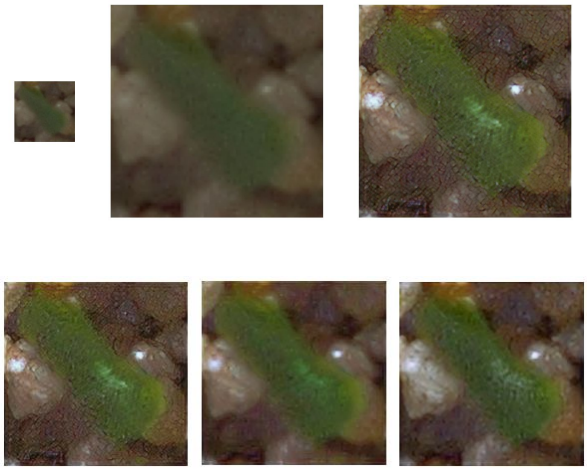

Figure 4. **Up:** The example of DCGAN, from left to right are original image, resize image and generated image; **Down:** The example of methods, from left to right are two different G*2 and G*4