Final Report for " Plant Pathology 2020 - FGVC7"

By Li Xinyan (55670594), Yu Yuxuan (55956283)

Project Group 48

1 Problem description

Our project uses the dataset is provided by Thapa, R. (2020). This data set is deemed to be used to classify the foliar disease of apples. The project is quite meaningful, for it will contribute to disease detection in the early stage and hence the disease in apple trees can be handled in the early stage. Detecting the disease as early as possible can not only increase apple production but also decrease toxic farm chemical usage for environmental benefits.

2 Hardwares and softwares used in experiment

In this project, the hardware uses the TitanXp 12G GPU, and the softwares are tensorflow 2.0.0, keras 2.3.1, sklearn 0.22.4

3 Literature Review

In the last decade, similar work has been done to detect or classify leaf diseases for various plants, such as banana, tomato, and so on. LeNet architecture as a convolutional neural network was used to classify banana sigatoka and banana speckle, achieving the accuracy of nearly 0.93 at least in the color situation (Amara, J. and et.al., 2017). In multiclass sugar leaf disease classification experiment, K-means clustering was used for segment and SVM classifier resulted in classification accuracy of around 93%.

ResNet is a deep learning architecture developed in 2015, its architecture is show in Figure 1 (He, K. et al, 2015). The architecture of MobileNet is shown in Figure 2 (Howard, A. G.,2017). That of Efficient B0 is shown in Figure 3, and Efficient B7, which is the renew version of B0, is used in our project(Tan, M., Le, Q. V., 2019).

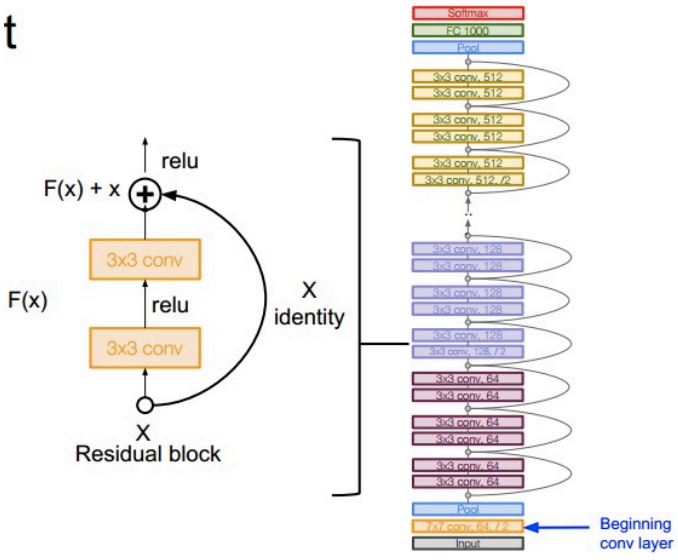


Figure 1 ResNet architectures

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2 MobileNet architectures

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $(\hat{H}_i, \hat{W}_i)$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3 EfficientNet-B0 architectures

4 Data Overview

4.1 Data summary

The train data contains 1801 pictures and the test one contains 1801 pictures too. For train dataset, the pictures include healthy leaves, multiple diseases leaves, rust leaves and scab

ones. The portions of different leaves are shown in Figure 4. The examples of different leaves are shown in Figure 5 to Figure 8.

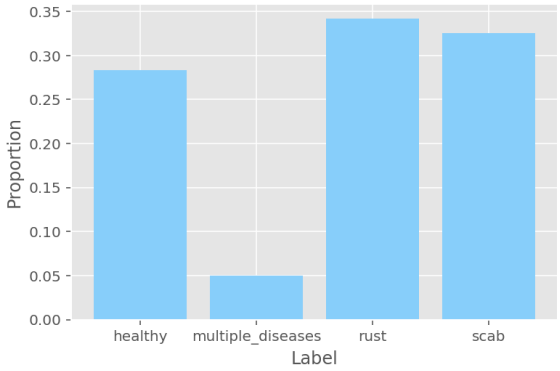


Figure 4 portions of different leaves

Figure 5 scab leaf

Figure 6 multiple diseases leaf

Figure 7 healthy leaf

Figure 8 rust leaf

4.2 RGB information of the images

RGB color model is an additive color model, including red, green and blue three primary color beams. They can be added together in various ways to reproduce numerous colors. The RGB models for different leaves are shown in Figure 9 to Figure 12.

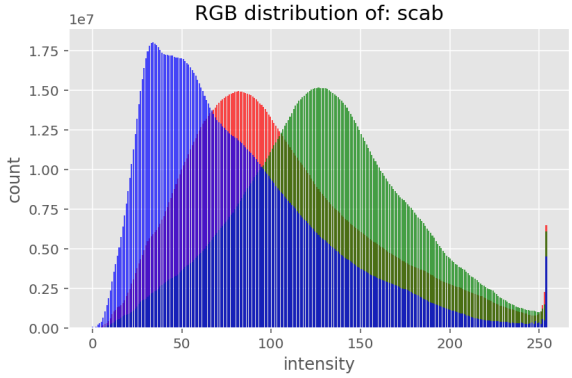


Figure 9 scab leaf

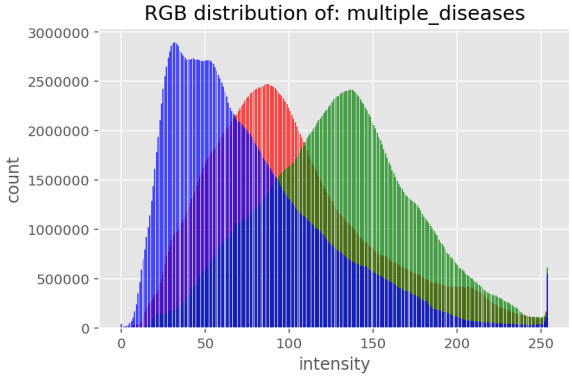


Figure 10 multiple diseases leaf

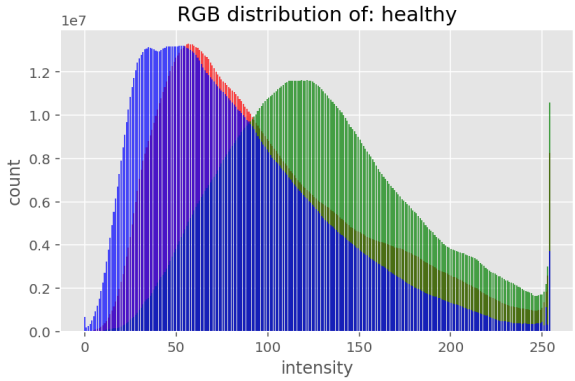


Figure 11 healthy leaf

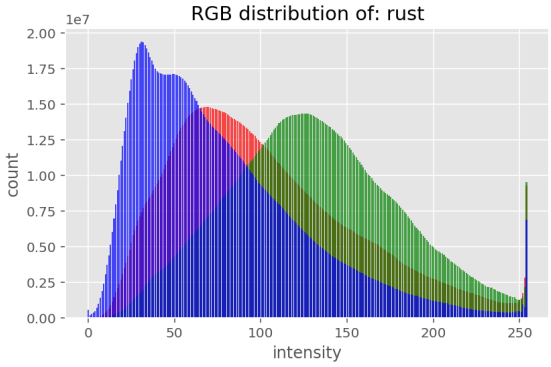


Figure 12 rust leaf

5 Baseline Experiments

5.1 Baseline Experiment Description

This part used the Principle component analysis (PCA) methods, with the 26 components, to fit and extract features from train dataset and test dataset. Then the train and test dataset are unit scaled. Then the data is classified with logistic regression multi-classifier and random forest classifier. Grid search and cross validation methods are used to pick up the best parameter for classification. With logistic regression classifier, the accuracy of 0.523 is achieved. And with random forest classifier, the accuracy of 0.535 is achieved. The running time for this experiment with nearly 10 minutes

The parameters used in random forest classifier are as follows: ***best params: {'max_depth': 5, 'min_samples_leaf': 0.04542495334260416, 'min_samples_split': 0.11946711107795627}.*** The logistic regression used the "***Limited-memory Broyden–Fletcher–Goldfarb–Shanno(lbfgs)***" optimizer.

5.2 Baseline Experiment Evaluation and Analysis

To evaluate the methods, the experiment is re-ran based on new train data, which contains 80% of the original train images as the train dataset, and 20% of the original train images as test dataset.

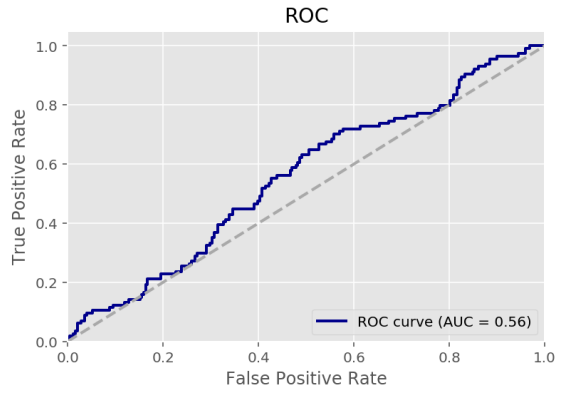


Figure 13 ROC curve for logistic regression classifier

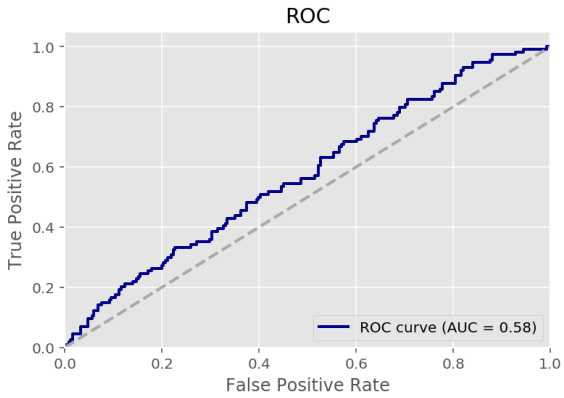


Figure 14 ROC curve for random forest classifier

PCA method may work fine in text classification, where the feature vector is quite sparse. For version classification, PCA cannot give enough information about for classification.

6 Advanced Experiment: Deep Learning methods

The accuracy of baseline experiments is too low to be helpful. Hence, we try deep learning methods.

6.1 Experiment 1: ResNet50 and SVM with RBF kernel

In experiment 1, ResNet50 is used for feature extraction, where pre-trained weights from ImageNet are used, and support vector method (SVM) with RBF kernel is used for multi-type classification. The accuracy achieved is only 0.551. The time taken is nearly 40 minutes, most of which is spent on SVM model training.

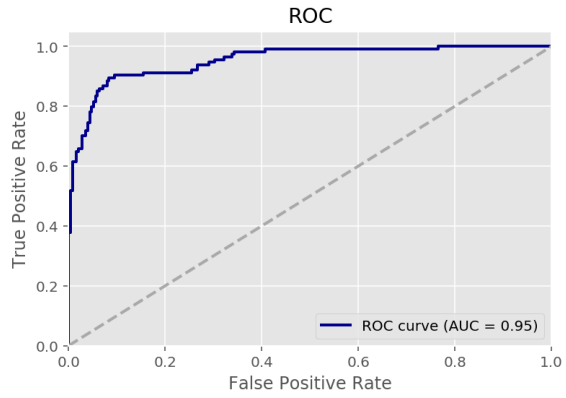


Figure 15 ROC curve for experiment 1

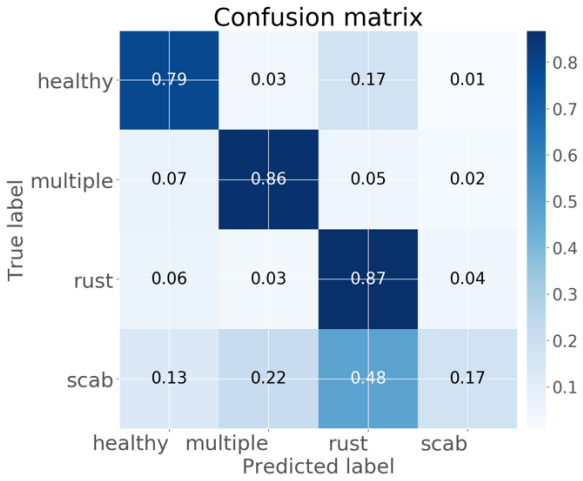


Figure 16 confusion matrix for experiment 1

As shown in the Figure 16, most of the misclassification comes from scab, it is usually misclassified into rust.

6.2 Experiment 2: ResNet50

In experiment 1, ResNet50 is used for classification, where pre-trained weights from ImageNet are used. The accuracy achieved is only 0.51. The time taken is nearly 80 minutes.

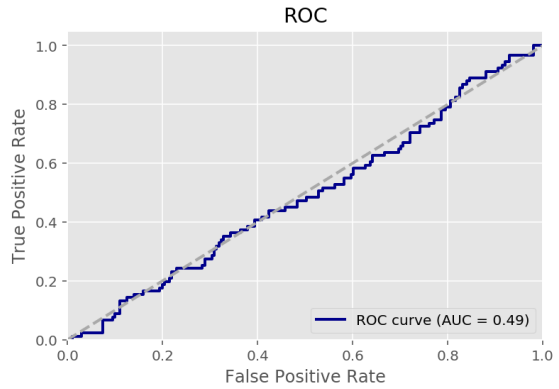


Figure 17 ROC curve for experiment 2

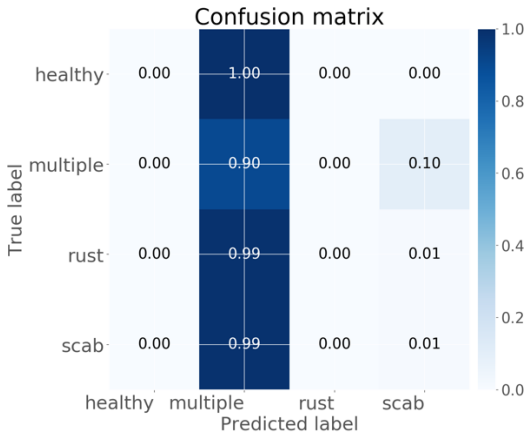


Figure 18 confusion matrix for experiment 2

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2048)              23587712
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 8196
=================================================================
Total params: 23,595,908
Trainable params: 8,196
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 19 model for experiment 2

6.3 Experiment 3: EfficientNetB7

In experiment 3, EfficientNetB7is used for classification, where pre-trained weights from ImageNet are used. The accuracy achieved is only 0.930. The time taken is nearly 10 hours. The model used is as follows.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b7 (Model)      (None, 7, 7, 2560)        64097680
_________________________________________________________________
global_average_pooling2d (Gl (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 4)                 10244
=================================================================
Total params: 64,107,924
Trainable params: 63,797,204
Non-trainable params: 310,720
_________________________________________________________________
```

No data augmentation is used in this experiment, for the limited of computation resources. The accuracy can be further improved if data augmentation is used. Because in this project, light, degree and the background can all influence the classification.

6.4 Experiment 4 for final submission:

The overall idea is to apply augmentation on training data, then use the augmented images to train a neural network.The input data was augmented with the help of albumentations library. The albumentations library provides interfaces for common image augmentation operations like blur, flip, rotate, scaling, resize, normalize and noise.
I tried three different sets of augmentation. They achieved accuracy of 0.937, 0.895 and 0. 500 at Kaagle submission page separately.

The first augmentation set is cited from the 1st place solution of *SIIM-ACR Pneumothorax Segmentation* (https://kaggle.com/c/siim-acr-pneumothorax-segmentation), the second and the third augmentation set is cited from the 1st and 7th place solution from *Planet: Understanding the Amazon from Space challenge* (https://www.kaggle.com/c/planet-understanding-the-amazon-from-space). I also made modification on the augmentation parameter to make it more personalized of my implementation. The code is show

```
aug_types = albu.Compose([

    albu.HorizontalFlip(),
    albu.OneOf([albu.HorizontalFlip(),albu.VerticalFlip(),],p=0.8),
    albu.OneOf([albu.RandomContrast(), albu.RandomGamma(), albu.RandomBrightness(),], p=0.3),
    albu.OneOf([albu.ElasticTransform(alpha=120, sigma=120 * 0.05, alpha_affine=120 * 0.03),
                albu.GridDistortion(), albu.OpticalDistortion(distort_limit=2, shift_limit=0.5),],
                p=0.3),
    albu.ShiftScaleRotate()
])

aug_types2 = albu.Compose([
    albu.RandomRotate90(p=0.5),
    albu.HorizontalFlip(p=0.5),
    albu.RandomGamma(gamma_limit=(80, 120), p=0.5),
    albu.JpegCompression(quality_lower=70, quality_upper=90, p=0.5),
    albu.RandomScale(scale_limit=(0.5, 2), interpolation=cv2.INTER_CUBIC, p=1),
    albu.Resize(224,224,always_apply=True),],
    p=1)

aug_types3 = albu.Compose([
    albu.HorizontalFlip(), albu.OneOf([albu.RandomContrast(),
                albu.RandomGamma(), albu.RandomBrightness(),], p=0.3),
    albu.OneOf([albu.ElasticTransform(alpha=120, sigma=120 * 0.05, alpha_affine=120 * 0.03),
                albu.GridDistortion(),
                albu.OpticalDistortion(distort_limit=2, shift_limit=0.5),],
                p=0.3),
    albu.ShiftScaleRotate(),
    albu.Resize(224,224,always_apply=True),    # a square image
])
```

Figure 20 Data augmentation for final submission

This is a model with very simple logic! Its structure is almost the same as the MobileNet. The reason why I adopted MobileNet is that it's a light convolution network and can be trained on mobile and embedded device. It can easily be trained on PC with CPU.
I used the pre-trained MobileNet except the last layer. For the last layer, I added a dense layer with softmax to make 4-class prediction.

To speed up the training process, I tested different initial value of learning rate and reduce the value dynamically when the network gets stable. Meanwhile, early stopping is introduced to prevent model from overfitting.

It takes about 4 hours to finish 50 epochs in training process (without early stopping). But the model actually achieved 0.9 accuracy at around 15 epochs (at around 10 epochs if increase the learning rate). Eventually, we only need less than 50 minutes to obtain a satisfying model (acc>0.9), and the time cost will be much lower with GPU enabled.

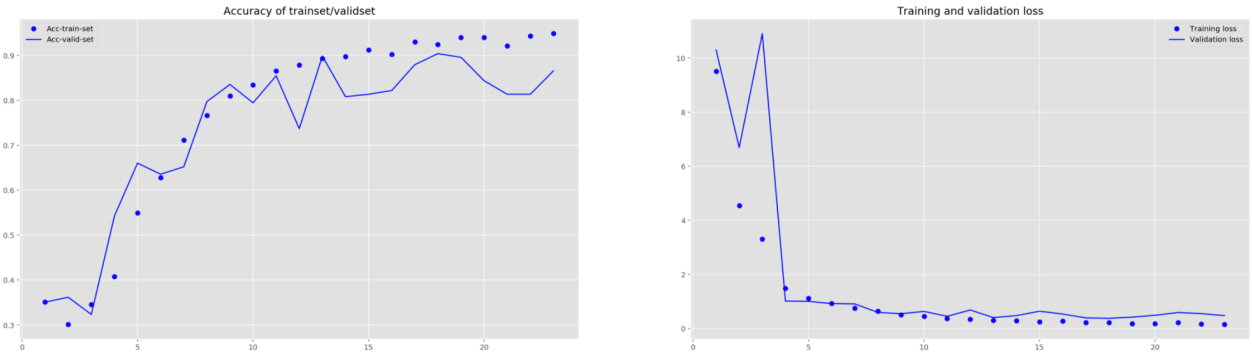
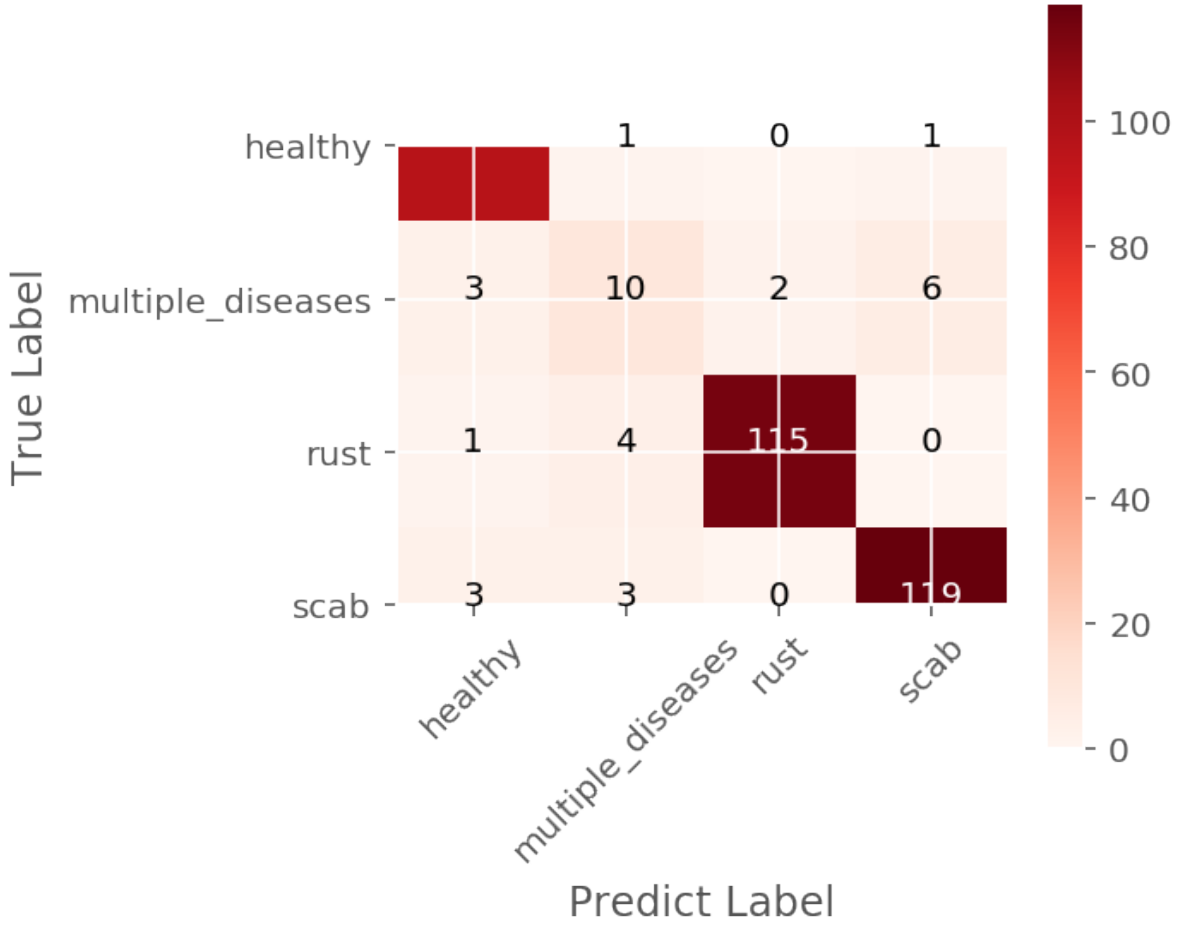


Figure 21 Model Performance for final submission



7 Conclusion and future work

In this project, it is quite computation resources consuming. In future work, it may be possible to use distribution computing method to reduce computation time. And the accuracy is expected to improve if we combine Efficient B7 and data augmentation method(blur, flip, rotate, scaling, resize, normalize and noise). Besides, background remove method is believed to help too. But all of above can be time consuming.

8 Team member contribution

Data overview: Li Xinyuan;

Baseline experiment: Yu Yuxuan;
Advanced experiment: Yuxuan, Li Xinyuan;
Report: Yuxuan, Li Xinyuan;
PPT & presentation: Yu Yuxuan

9 Reference

[1] Thapa, R., Snavely, N., Belongie, S., & Khan, A. (2020). The Plant Pathology 2020 challenge dataset to classify foliar disease of apples. *arXiv preprint arXiv:2004.11958*.
[2] Amara, J., Bouaziz, B., & Algergawy, A. (2017). A deep learning-based approach for banana leaf diseases classification. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband*.
[3] Rumpf, T., Mahlein, A. K., Steiner, U., Oerke, E. C., Dehne, H. W., & Plümer, L. (2010). Early detection and classification of plant diseases with support vector machines based on hyperspectral reflectance. *Computers and electronics in agriculture*, *74*(1), 91-99.
[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
[6] Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.