Image Classification of Plant Pathology 2021

Jinyi Shang

Yu Cao

The George Washington University

Contents

1. Introduction

Apples are one of the most important temperate fruit crops in the world. They contain a lot of vitamin C. It can improve the body's immunity, as well as reduce blood pressure, if people eat it regularly. As a result, apples have become a daily staple for many people. This not only provides huge benefits for consumers, but also for growers. However, foliar (leaf) diseases pose a major threat to the overall productivity and quality of apple orchards. This has the potential to cause huge losses for growers.

The U.S. apple industry, annually worth $15 billion, experiences millions of dollars in annual losses due to various biotic and abiotic stresses, ongoing stress management, and multi - year impacts from the loss of fruit - bearing trees. Over the growing season, apple orchards are under constant threat from a large number of insects, as well as fungal, bacterial, and viral pathogens, particularly in the northeastern United States. Depending on the incidence and severity of infection by diseases and insects, impacts range from unappealing cosmetic appearance, low marketability, and poor quality of fruit, to decreased yield or complete loss of fruit or trees, causing huge economic losses. Early pest and disease detection are critical for appropriate and timely deployment of disease and pest management programs.

Currently, disease and pest detection in commercial apple orchards relies on manual scouting by crop consultants and service providers. Unfortunately, very few experienced scouts are available, forcing them to cover many large orchards within a narrow time frame. Scouts require a great deal of expertise and training before they can be efficient and accurate in diagnosing an orchard. Generally, they are first trained using images of disease symptoms and insect damage, but due to the presence of a great number of variables in an actual orchard, they need considerable time to familiarize themselves with the many symptom classes caused by either the age and type of infected tissues or the stage of the disease or pest cycle, as well as by changing weather, geographical variances, and cultural differences. Many symptoms of diseases, pests, and abiotic stresses in an apple orchard are distinct enough to differentiate based on visual symptoms alone. However, several disease symptoms look similar enough to each other that it is difficult to accurately determine their cause. At the same time, visual symptoms of a single disease or particular insect can vary greatly between apple varieties, due to differences in leaf color, morphology, and physiology. In addition to the time spent in the orchard, a scout spends a significant amount of time on each client, entering the scouting report, interpreting results, and providing recommendations for action. Overall, human scouting is usually time consuming, expensive, and in some cases, prone to errors.

In recent years, digital imaging and machine learning have shown great potential to speed up plant disease diagnosis. In this report, we use deep learning to train models to categorize apple plant pathology categories. Transfer learning method is selected here. We apply pre-trained model approach to use parts of the pre-trained model. The models include VGG model and ResNet model in this research. These two models are developed by a challenging image classification task, ImageNet 1000-class photograph classification competition. These models can take days or weeks to train on modern hardware. They can be downloaded and incorporated directly into new models that expect image data as input. In order to improve the performance of models, some approaches are applied, such as deleting duplicates, encoding, transforming, image augmentation and so on.

2. Description of Dataset

2.1 Overview of the dataset

We obtained data from Plant Pathology 2021-FGVC8 challenge competition in Kaggle. Plant Pathology 2020-FGVC7 challenge competition had a pilot dataset of 3,651 RGB images of foliar disease of apples. For Plant Pathology 2021-FGVC8, dataset has significantly increased the number of foliar disease images and added additional disease categories. The dataset contains approximately 23,000 high-quality RGB images of apple foliar diseases, including a large expert-annotated disease dataset. This dataset reflects real field scenarios by representing non-homogeneous backgrounds of leaf images taken at different maturity stages and at different times of day under different focal camera settings.

The train.csv file contains two columns, image and labels. Image indicates the image ID and labels represents the target classes, a space delimited list of all diseases found in the image.

The original train.csv file contains 18632 images. The detailed number of each disease is listed in Table 1.

Table 1. Counts of Each Disease

Disease	**Counts**
scab	4826
healthy	4624
frog_eye_leaf_spot	3181
rust	1860
complex	1602
powdery_mildew	1184
scab frog_eye_leaf_spot	686
scab frog_eye_leaf_spot complex	200
frog_eye_leaf_spot complex	165
rust frog_eye_leaf_spot	120
rust complex	97

powdery_mildew complex	87

2.2 Exploratory Data Analysis

2.2.1 Duplicates Values

Duplicates are always harmful for training process: differently labeled duplicates produce noise in the dataset, while equally labeled duplicates lead to data leakage. Duplicate data will affect the statistical results and mislead decision makers. In the dataset, there are more than 50 duplicates.

In this report, we used the result of a notebook in Kaggle to solve duplicates (https://www.kaggle.com/nickuzmenkov/pp2021-duplicates-revealing/output). In his notebook, firstly he saved downscaled images to boost performance because of long time taken by computing hash over original images of very high quality. Then, he used hash to find duplicated images and save them into a csv file. We use this csv file directly to get a dataset without duplicates.

2.2.2 Encoder

In this report, we take different targets as different labels instead of using multi-labels. That is label 'complex' as one category, label 'frog_eye_leaf_spot' as one category, and label 'frog_eye_leaf_spot complex' as one category. So, there are total of 12 categories in the dataset, including 'complex', 'frog_eye_leaf_spot', 'frog_eye_leaf_spot complex', 'healthy', 'powdery_mildew', 'powdery_mildew complex', 'rust', 'rust complex', 'rust frog_eye_leaf_spot', 'scab', 'scab frog_eye_leaf_spot', 'scab frog_eye_leaf_spot complex'. Thus, the labels are encoded into 0 to 11 depending on different categories.

2.3 Data Expansion

Since our data set is imbalance and to solve this imbalance problem, the most straight forward consideration we think is using data expansion method.

Here, we use a package called albumentaions to achieve this goal. Albumentations is a Python library for fast and flexible image augmentations. Albumentations efficiently implements a rich variety of image transform operations that are optimized for performance, and does so while providing a concise, yet powerful image augmentation interface for different computer vision tasks, including object classification, segmentation, and detection. In this package, there are mainly two different image transformation: the pixel_level transform and the spatial_level transform.

Pixel-level transforms will change just an input image and will leave any additional targets such as masks, bounding boxes, and keypoints unchanged. Usually PLT will add noise to the original image. Spatial-level transforms will simultaneously change both an input image as well as additional targets such as masks, bounding boxes, and keypoints. Usually SLT will not add noise to the original image.

To solve this imbalance problem, we build two different expanded dataset based on different strategies: the first one, we combine the pixel_level transform and the spatial_level transform together to generate our new image and the second one we only use spatial_level transform.

The reason we build two expanded dataset is for the first dataset, we applied many different pixel_level transform, which may be good for preventing overfitting, but it may cause too much noises added to original image as well. To eliminate this possibility, we adjust the porpotion of pixel_level transform we use in our second expanded dataset and apply both expanded dataset to our model to check which is better.

Finally, after data expansion, we expand the amount of our image data from 14000 to 44000 and 39000, separately.

3. Model and Algorithm

3.2 ResNet Model

The proposal of Deep Residual Network (ResNet) was a milestone in the history of CNN images.

From experience, the depth of the network is crucial to the performance of the model. When the number of network layers is increased, the network can carry out more complex feature pattern extraction, so better results can be obtained theoretically when the model is deeper.

The experiment found that the depth network presented a degradation problem: with the increase of the depth of the network, the network accuracy appeared saturation, or even decreased. The problem is clearly not caused by overfitting, because not only does the test error become higher as the network deepens, so does its training error. This can be seen from Figure.

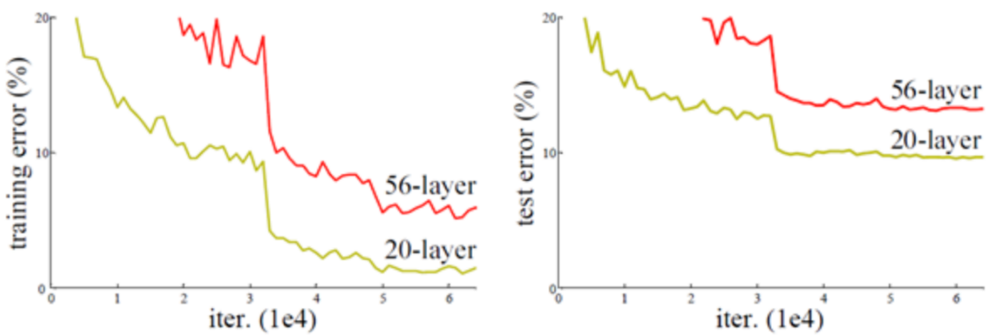


Figure . Degradation Problem

When the number of layers of the traditional neural network increases from 20 to 56, both the training error and the test error of the network increase significantly, that is to say, the performance of the network degenerates significantly with the increase of the depth. ResNet was created to solve this degradation problem.

Residual learning is proposed to solve the degradation problem. For a stacking layer structure, when the input is x, the feature learned is noted as $H(x)$. Now we hope that it can learn residual $F(x) = H(x) - x$, so that the original learning feature is actually $F(x) + x$. The reason for this is that it is easier to learn residuals than to learn the original features directly. When the residual is 0, the stack layer only does the identity mapping, and at least the network performance will not decrease. In fact, the residual will not be 0, which also makes the stack layer learn new features on the basis of input features, so as to have better performance. The structure of residual learning is shown in Figure. This is a kind of "short" in an electric circuit, so it's a shortcut connection.

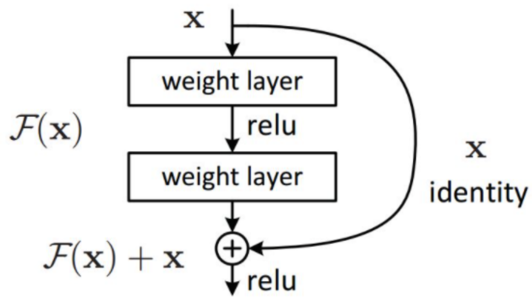


Figure . The Basic Unit of Residual Study

4. Experimental Setup

4.1 Create data pipeline

In this project, we mainly use pytorch to execute our model VGG16 and Resnet50. Before we configure our model, we need to create the data pipeline which is available to feed data to our model.

4.1.1 Transform class setup

Data does not always come in its final processed form that is required for training machine learning algorithms. Therefore, We use transforms to perform some manipulation of the data and make it suitable for training.

Here's the snippet of our transform class code:

```
class pl_transform():
    def __init__(self):
        self.plant_transform = {
            'train':transforms.Compose([
                transforms.RandomResizedCrop(
                    224, scale=(0.5, 1.0)),
#                 transforms.RandomHorizontalFlip(),
#                 transforms.RandomVerticalFlip(0.2)
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0
```

we can add different transformation here, for example, VGG16 requires the input image size as 224 by 224 and normalize the image with mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]. what's more, we can use data augmentation techniques here, like the annotate snippet part shows, we can add random horizontal flip or random vertical flip and other transforms here to prevent overfitting.

4.1.1 Customized dataset class setup

At the heart of PyTorch data loading utility is the 'torch.utils.data.DataLoader' class. It represents a Python iterable over a dataset and The most important argument of DataLoader constructor is dataset, which indicates a dataset object to load data from.

Since data loading order is entirely controlled by the user-defined iterable.We can easily define our own dataset shown below and this allows easier implementations of chunk-reading and dynamic batch size:

```
class mydataset(Dataset):
    def __init__(self , csv_file , img_dir , transforms=None, phase = 'train' ):
        self.targetfile = csv_file
        self.root = img_dir
        self.transforms = transforms
        self.phase = phase


    def __len__(self):
        return len(self.targetfile)

    def __getitem__(self, idx):
        img_path = os.path.join(self.root,self.targetfile.iloc[idx,0])
        image = Image.open(img_path)
        label = self.targetfile.iloc[idx,1]

        if self.transforms:
            image = self.transforms(image,self.phase)
        return image,label
```

As we can see, all we need to define is a __init__ function for initialization , __len__ function to return a customized length of dataset and __getitem__ function to retrieve the data.

After we defined our dataset, all we need to do is feed our dataset to a dataloader like this:

```
train_loader = DataLoader(train_dataset,
#                          sampler = wsampler,
                            batch_size = BATCH_SIZE,
                            shuffle = True,
                           )
```

Quite simple, right? At this point, we have completed all the data pipeline configuration.

4.2 Model Configuration

Since we don't need to build our own model, it's quite simple to load VGG16 and Resnet50 to our code like this:

```
def model_define():
    use_pretrained = True
    Mymodel = models.resnet18(pretrained=use_pretrained)
    Mymodel.fc = nn.Sequential(
#             nn.Linear(2048, 1024),
#             nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 12)
        )
```

As you can see, all we need to do is to change the last fully connected layers to make it match our expected output.

Then, we use fine-tuning strategy to train our model. Fine-tuning means taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain (often e.g. images). It is used to speed up the training or overcome small dataset size.

Here, for each of model, we will "freezing" some of the pre-trained weights (usually whole layers) and train and update only a part of the parameters of the network.

Take Resnet as example:

```
Parameters not to be learned :  layer4.0.bn1.bias
Parameters not to be learned :  layer4.0.conv2.weight
Parameters not to be learned :  layer4.0.bn2.weight
Parameters not to be learned :  layer4.0.bn2.bias
Parameters not to be learned :  layer4.0.downsample.0.weight
Parameters not to be learned :  layer4.0.downsample.1.weight
Parameters not to be learned :  layer4.0.downsample.1.bias
Store in params_to_update_1 : layer4.1.conv1.weight
Store in params_to_update_1 : layer4.1.bn1.weight
Store in params_to_update_1 : layer4.1.bn1.bias
Store in params_to_update_1 : layer4.1.conv2.weight
Store in params_to_update_1 : layer4.1.bn2.weight
Store in params_to_update_1 : layer4.1.bn2.bias
Store in params_to_update_1 : fc.1.weight
Store in params_to_update_1 : fc.1.bias
```

From the printout information, it shows that from layers 1.0 to layers 4.0, all of the parameter will be freezed and in the training process, we will only update the parameters from layers 4.1 to the final fully connected layers.

After we finished our training processing, we will combine these two models' results and feed the results to a ensemble classifier and obtain our final prediction.

4.3 Metrics

4.4 About hyper-parameter

Here's the hyper-parameter we used:

```
SEED = 42
EPOCHS = 6
LR = 1e-5
MIN_LR = 1e-7
MODE = 'min'
FACTOR = 0.2
PATIENCE = 0
BATCH_SIZE = 128
TEST_SIZE = 0.2
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

For the epochs selection, we use early stopping strategy, we can just plot the train loss curve and validation curve, then we can decide a suitable epochs number. Here's a snippet of our plot code:

```
fig = plt.figure(figsize=(7, 6))
plt.grid(True)
plt.plot(train_acc, color='r',marker='o', label='train/acc')
plt.plot(val_acc, color='b',marker='x',label='val/acc')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(loc='lower right')
plt.show()
```

For the learning rate selection

For the mini-batch size selection, In general, larger batch sizes result in faster progress in training, but don't always converge as fast. Smaller batch sizes train slower, but can converge faster. Therefore, we set our batch size as 32, 64, 128 and 256 and to check the performance.

The result shows in this dataset, 128 is a better choice.

4.4 About over-fitting

For the over-fitting problem, the mainly strategy is in transform part, we add many pixel_level transforms, include: Blur, ColorJitter, CLAHE, FancyPCA, RandomSunFlare, RandomFog and so on. Using this methods, we can add noises manually to the original image, which is useful to prevent over-fitting.

The second strategy we use is increasing the rate of dropout layer in our model. As we know dropout layer is good for preventing over-fitting.

The third strategy, we decrease the model complexity, using Resnet18 as substitute.

5. Results

5.1 VGG16

Here's the VGG16 result:

```
val Loss: 1.2190 Acc: 0.5963
The f1 score is 0.6056423774343158
```

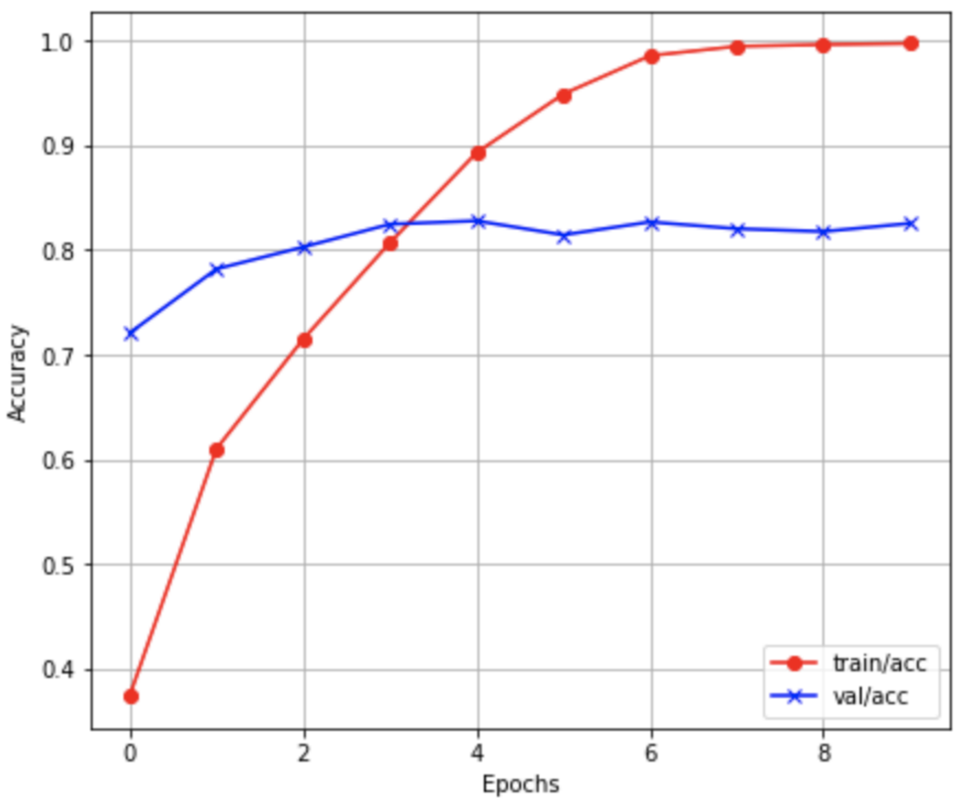
vgg_sub
(version 6/6)
a day ago by Yu Cao
From Notebook [vgg_sub]

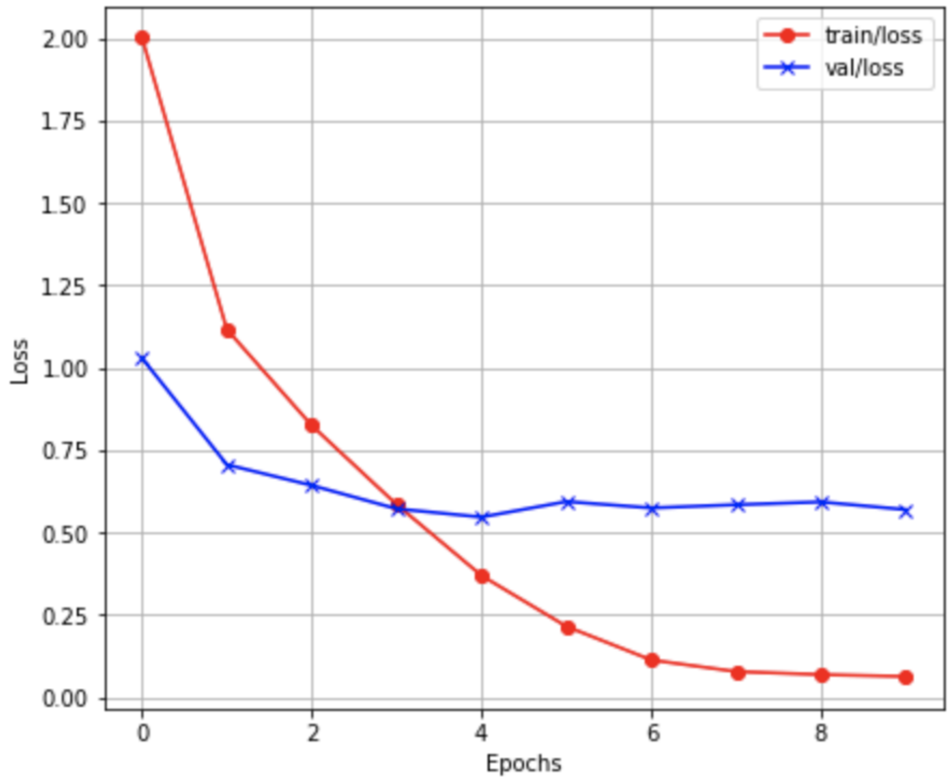
Succeeded 0.640

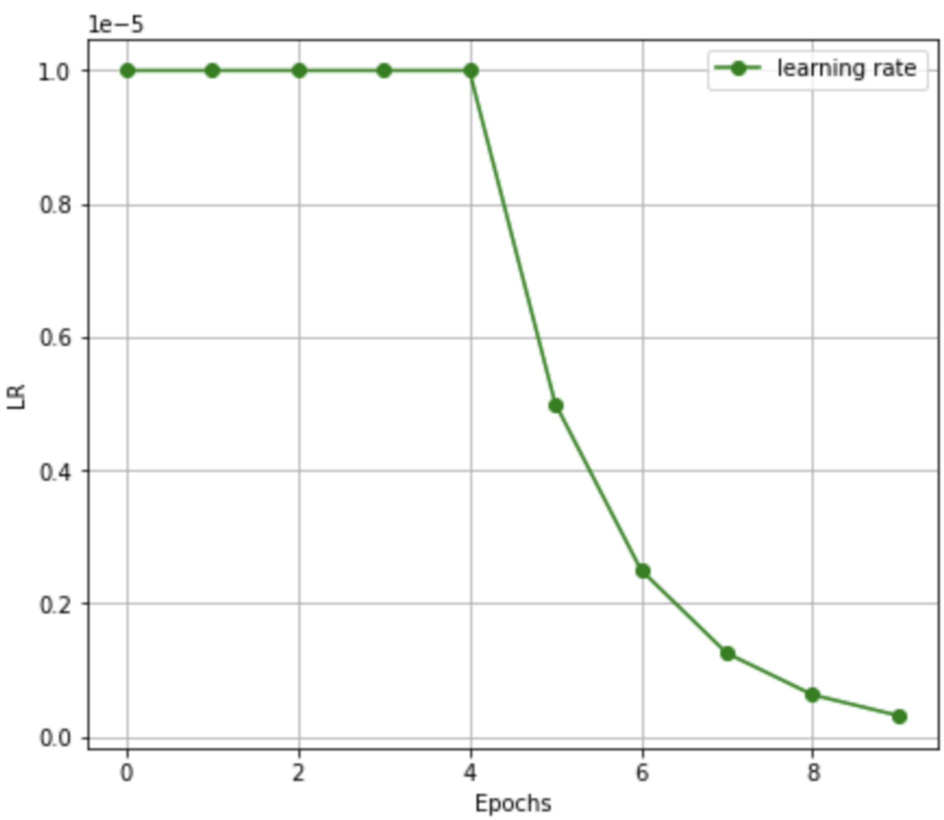
As we can see, the local f1 score is 0.6056 and the leader board f1 score on kaggle is 0.64. The LB score is better than Local, which is a good news.

5.2 Resnet50

For the Resnet50 results, we plot the train_val accuracy, train_val loss and learning rate decay graphs and here are the results:




train/loss
val/loss
Loss
Epochs
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8


1e−5
learning rate
LR
Epochs
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

For the local result:

```
val Loss: 0.5716 Acc: 0.8248
The f1 score is 0.8215232721433382
```

As you can see, the local loss and local f1 score is really good, the f1 score reach 0.8, but on the leader board:

fi_ressub (version 6/9) a day ago by Yu Cao18 From Notebook [fi_ressub]	Succeeded	0.257

For the second expanded dataset result:

```
val Loss: 1.1496 Acc: 0.6128
The f1 score is 0.6497946916085479
```

ressub (version 3/4) 9 hours ago by Jinyi Shang	Succeeded	0.197

We think it may be a transform problem,so, after we rebuild our dataset and run it again, the result gets worse. It may be not the transformation problem.

For the third result, we have used the Resnet18 as a substitute to decrease the data complexity and here it is:

```
val Loss: 1.3389 Acc: 0.5721
The f1 score is 0.6004212701820477
```

ressub (version 4/4) 7 hours ago by Jinyi Shang	Succeeded	0.119

The result is still bad and we didn't figure it out yet.

5.3 Ensemble result

In our schedule, we plan to use ensemble methods to combine the result of our model and here's the result:

```
f1_score(pred,y_val,average='micro')
```

```
0.7190860215053764
```

failed_ensemble
(version 2/2)
a day ago by Jinyi Shang
From Notebook [failed_ensemble]

Succeeded 0.603

On the local side, the f1 score is 0.71 and on leader board it is 0.6. Since the performance of resnet is still bad, I think this result is reasonable(a good model and a really bad one will not generate a better one.)

6. Summary and Conclusions

In this project, I think I have a comprehensive understanding of how to build a deep neural network, include how to define transform class, how to define dataset class and dataloader and how to use transfer learning using pytorch. what's more, this is the really interesting competition that I can handle some really good data. It gives me a lot.

In our final result, using single VGG16 obtains the best, but we certainly sure our model can be better if we can solve the problem of our resnet50 model and the potential problem we think are:

1, Too much noise

2, over-fitting

3, data distribution is different between train and test set

4, validation loss is smaller than train loss

In the future, except to solve the Resnet50's problem, we should also focus more on data preprocessing, like using suitable transformation to make the image easier to be recognized, adjust the image size more suitable and so on. What's more, we can solve this problem by treating this problem as a multilabel problem and using BCEloss() to deal with it. I think if we can combine more suitable models together, we can decrease the high variance of neural network and have a significant improvement.

7. References

[1] “Plant Pathology 2021 - FGVC8.” Kaggle, www.kaggle.com/c/plant-pathology-2021-fgvc8.

[2] Brownlee, Jason. “A Gentle Introduction to Transfer Learning for Deep Learning.” Machine Learning Mastery, 16 Sept. 2019, machinelearningmastery.com/transfer-learning-for-deep-learning/#:~:text=Transfer learning is a machine,model on a second task.&text=Common examples of transfer learning,your own predictive modeling problems.

[3] “13.2. Fine-Tuning¶ Colab [Mxnet] Open the Notebook in Colab Colab [Pytorch] Open the Notebook in Colab Colab [Tensorflow] Open the Notebook in Colab.” 13.2. Fine-Tuning - Dive into Deep Learning 0.16.3 Documentation, d2l.ai/chapter_computer-vision/fine-tuning.html.

[4] Huilgol, Purva. “Accuracy vs. F1-Score.” Medium, Analytics Vidhya, 24 Aug. 2019, medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2.

[5] Nickuzmenkov. “PP2021 - Duplicates Revealing.” Kaggle, Kaggle, 6 Apr. 2021, www.kaggle.com/nickuzmenkov/pp2021-duplicates-revealing.

[6] Kuboko. “PP2021_PyTorch_VGG-16_Fine-tune_ Inference.” Kaggle, Kaggle, 25 Mar. 2021, www.kaggle.com/kuboko/pp2021-pytorch-vgg-16-fine-tune-inference.

[7] Thapa, R., Zhang, K., Snavely, N., Belongie, S., & Khan, A. (2020). The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences, 8(9). doi:10.1002/aps3.11390

[8] Compendium of Apple and Pear Diseases and Pests, Second Edition. (2014). doi:10.1094/9780890544334

[9] Bessin, R., P. McManus, G. Brown, and J. Strang. 1998. Midwest tree fruit pest management handbook. University of Kentucky, Lexington, Kentucky, USA

[10] Judd, G. J., A. L. Knight, and A. M. El‐Sayed. 2017. Development of kairomone‐based lures and traps targeting Spilonota ocellana (Lepidoptera: Tortricidae) in apple orchards treated with sex pheromones. Canadian Entomologist 149: 662–676.

[11] Li, X., S. Geng, H. Chen, C. Jung, C. Wang, H. Tu, and J. Zhang. 2017. Mass trapping of apple leafminer, Phyllonorycter ringoniella with sex pheromone traps in apple orchards. Journal of Asia‐Pacific Entomology 20: 43–46.

[12] Barbedo, J. G. A. 2014. An automatic method to detect and measure leaf

disease symptoms using digital image processing. Plant Disease 98: 1709–1716.

[13] Dai, P., X. Liang, Y. Wang, M. L. Gleason, R. Zhang, and G. Sun. 2019. High humidity and age - dependent fruit susceptibility promote development of Trichothecium black spot on apple. Plant Disease 103: 259–267.

[14] Mahlein, A. - K. 2016. Plant disease detection by imaging sensors: Parallels and specific demands for precision agriculture and plant phenotyping. Plant Disease 100: 241–251.

[15] Praveengovi. (2021, April 19). Plant Pathology-Detail-EDA Pytorch. Retrieved from https://www.kaggle.com/praveengovi/plant-pathology-detail-eda-pytorch

Appendix