Plant Pathology 2020 – FGVC7

——Identify the category of foliar diseases in apple trees

Kang Huang

1.Introduction

Misdiagnosis of the many diseases impacting agricultural crops can lead to many serious consequences such as misusing of chemicals leading to the emergence of resistant pathogen strains, increasing input costs and more outbreaks with significant economic loss and environment impacts. Current diseases diagnosis based on human scouting is time-consuming and expensive. With the increase of computing power, especially in the past decade, we can build deep learning models to help us diagnose. The computer-vision based on deep learning models have the promise to increase efficiency. However, the great variance in symptoms due to age of infected tissues, genetic variations, and light conditions within trees might increase the complexity of this problem and decreases the accuracy of detection.

The purpose of our project is to identify the category of foliar diseases in apple trees. We participated in a competition from Kaggle where we downloaded the datasets. It includes 1821 train images and 1821 test images. There are four targets: healthy, multiple_diseases, rust and scab. And it is only one category for each image. Before we start building models, we do some exploratory data analysis (EDA) of datasets. Through EDA, we find there are some problems of metadata such as imbalanced classes, inconsistent image size and duplicated data. To deal with these problems, we use different preprocessing methods, for example, resize, flipping, rotation to augment the data. After data preprocessing, we briefly introduce some essential theories and algorithms that we use in our model. Then, we show the architecture of our model and the process of tuning hyperparameters. Finally, we show performance of our model and summarize the results we obtained.

2. Individual Work

In the project, I mainly work on the theory introduction part. And all of us work on the model. Then we compare our models and choose the best one. So in this part, there is no EDA.
We use CNN to do the project, so I mainly talk about the theory of CNN. Convolution neural networks are a specialized kind of neural network, most commonly applied to analyzing visual imagery. Examples include time-series data, which can be thought of as a 1D grid taking

samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

In its most general form, convolution is an operation on two functions of a real-valued argument. To explain what convolution is, we provide an example. $x(t)$ is the location of the spaceship at time t provided by laser sensor. We suppose that the laser sensor is somewhat noisy. In order to get a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function w(a), where a is the age of a measurement. If we apply such a weighted average operation at every moment, we obtain a new function s providing a smoothed estimate of the position of the spaceship:

$$s(t) = \int x(a)w(t-a)da$$

This operation is called convolution. We can also denote it as $s(t) = (x * w)(t)$. In convolution network, the convolution layer is the principal layer. In this layer, x in the last equation means input and w refers to kernel. The output s sometimes refers to feature map. For example, let the image input be a $R_r \times R_c$ matrix V. And kernel is represented by a $r \times c$ matrix W. Then the output is:

$$z_{ij} = \sum_{k=1}^{r} \sum_{1=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

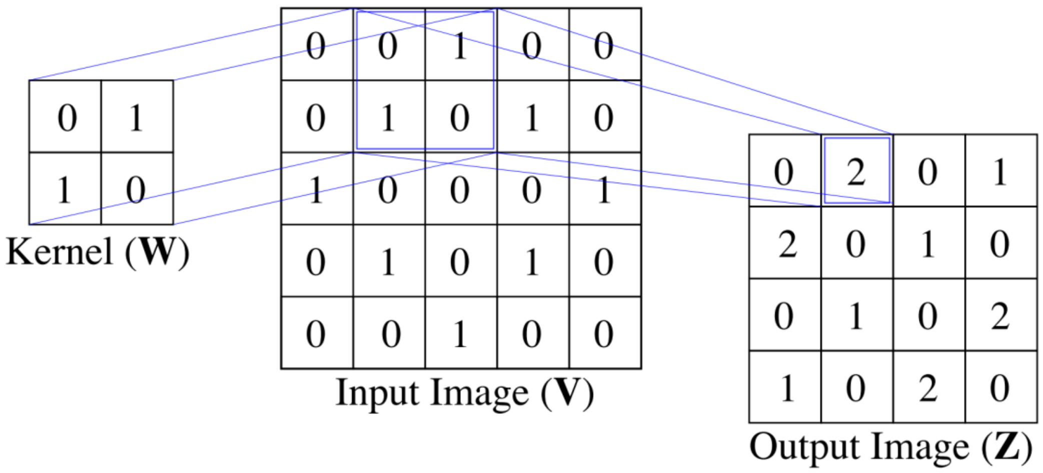


Figure 1 convolution operation

In addition to convolution layer, convolution neural networks always have another layer: sample layer. In this layer we use pooling function. The pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For instance, the max pooling operation reports the maximum output within a rectangular neighborhood.

$$z_{i,j} = \max\{v_{r(i-1)+k,c(j-1)+l} | k = 1, \dots, r; l = 1, \dots, c\}$$

In general, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

We choose this network to do our image classification because convolution networks can help improve a machine learning system in three ways: sparse interactions, parameter sharing and equivariant representations. Convolution networks usually have sparse interactions. Just as the figure shows, the output image is smaller than the input because of the convolution operation. This really This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large. Parameter sharing refers to using the same parameter for more than one function in a model. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. It can reduce the storage of the model. And parameter sharing causes the layer to have a property called equivariance to translation. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations.

And how to train this model? Here we also use forward pass to get the output and use back propagation to update the weights. And we validate this updated model to get roc auc score. If we get a higher score, then we save this model. Thus, we can have a best validation score.

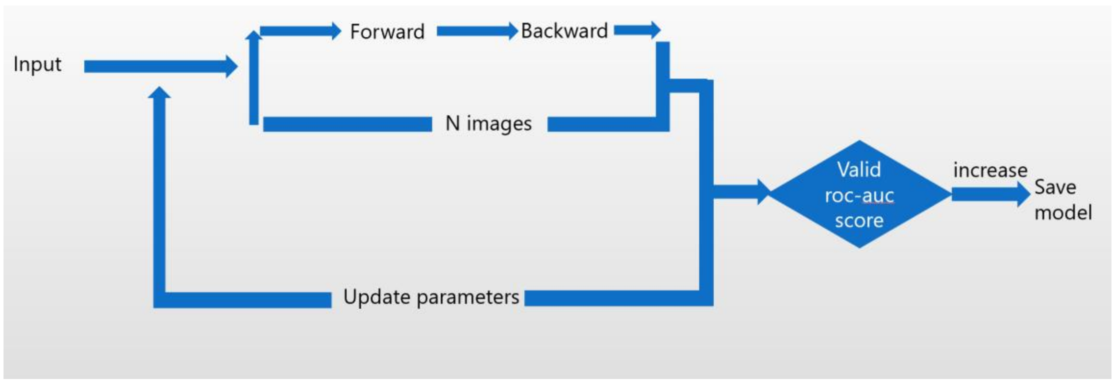


Figure 2 training algorithm

3. Model

In addition to the theory part, I also train the model for the project. Because all of us build models for the project, I want to be different in order to compare. So, I choose a pretrained model. This pretrained model is resnet18, from torchvision package.

To build a model, we should do the data preprocessing first. To load images from the dataset,

I create a function called PlantDataset to load images and labels from the dataframe so that I could use torchvision.dataloader function to load images and labels. There is another advantage for this function: we can do the augmentation in this function as well as loading images and labels.
And for the data augmentation, I use package albumentations as A. Here is what I do:

```
#augmentation
transforms_train = A.Compose([
    A.RandomResizedCrop(height=SIZE, width=SIZE, p=1.0),
    A.Flip(),
    A.ShiftScaleRotate(rotate_limit=1.0, p=0.8),

    # Pixels
    A.OneOf([
        A.IAAEmboss(p=1.0),
        A.IAASharpen(p=1.0),
        A.Blur(p=1.0),
    ], p=0.5),

    # Affine
    A.OneOf([
        A.ElasticTransform(p=1.0),
        A.IAAPiecewiseAffine(p=1.0)
    ], p=0.5),

    A.Normalize(p=1.0),
    ToTensorV2(p=1.0),
])

transforms_valid = A.Compose([
    A.Resize(height=SIZE, width=SIZE, p=1.0),
    A.Normalize(p=1.0),
    ToTensorV2(p=1.0),
])
```

And for the criterion, I create a function, which is similar to crossentropy in keras. Because in pytorch, if we use crossentropy, the category must be coded in 0,1,2,3, not one-hot encoding. And in this project, the labels are one-hot encoding. So we can not the original loss function crossentropy. Here is the function:

```
class DenseCrossEntropy(nn.Module):

    def __init__(self):
        super(DenseCrossEntropy, self).__init__()
    def forward(self, logits, labels):
        logits = logits.float()
        labels = labels.float()

        logprobs = F.log_softmax(logits, dim=-1)

        loss = -labels * logprobs
        loss = loss.sum(-1)

        return loss.mean()
```

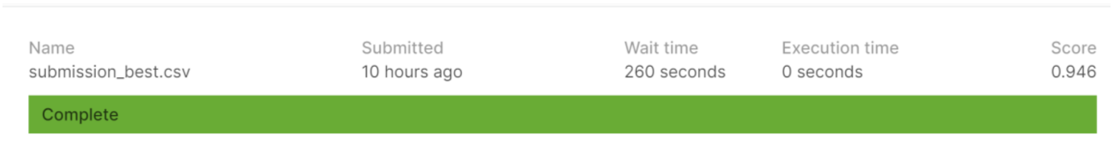
And in the training, because it is an imbalanced dataset (shown in the plot), I use stratified K-folds to improve roc auc score. The number of epochs are small, so I do not set a early stopping point. I just save the model when the validation score is higher than before.

Finally, I do the hyperparameter tuning. In this part, there are several hyperparameters. I have tried many times, and this is what I think is very suitable for the model. And I choose Adam as the optimizer.

```
N_FOLDS = 2
BATCH_SIZE = 64
SIZE = 512
LR=5e-5
```

4. Results

Using the model that I save in the training part, I can get the test prediction. Then I submit it to the Kaggle, the final score is 0.946. The rank is about 500.



Name	Submitted	Wait time	Execution time	Score
submission_best.csv	10 hours ago	260 seconds	0 seconds	0.946

Complete

Figure 3 submission score

And I also summarize validation score when using different learning rate.

Learning rate	Valid score
0.1	0.59243
0.01	0.58913
0.001	0.53273
0.0001	0.98644

0.00005	0.98173

Table 1 learning rate vs valid score

I submitted the prediction of lr=0.0001 and lr=0.00005. The score of lr=0.0001 is 0.931, which is lower. So I choose 0.00005 as my learning rate.

5. summary and conclusion

The result reveals that the CNN is quite suitable for the data. The roc auc score is 0.946, which is not bad I think. So for the image classification problem, I think CNN is a good choice.
And when doing the project, I think data preprocessing is particularly important. When I first built the model, I did not use the package albumentations and torchvision.dataloader function. I use skimage to transform the image and do the augmentation. But the result is not good. The validation score is only about 0.7. So I have to change the form of data preprocess. And as the table shown above, the hyperparameter tuning is also significant. The learning rate can make a big difference on the validation score. And when dealing with the imbalanced data, I find the stratified k-folds is a very good way, which I may ignore in the last 2 exams.
After doing this project, what impresses me most is that efficiency is important as well as accuracy. It is not easy to balacnce efficiency and accuracy. In this project, when doing hyperparamter tuning, if we use many epochs, it may take a long time. So we have to choose to lose some efficiency to speed up the process by using less epochs. To balance efficiency and accuracy, it is a good way to try different ways or change some parameters. We should get more experience dealing with the efficiency and accuracy.

The percentage of the code that I found or copied from the internet: 40.44%

Reference

1. Kaggle notebook: https://www.kaggle.com/pestipeti/plant-pathology-2020-pytorch
2. Pytorch tutorial: https://pytorch.org/tutorials/
3. Liu, T., Fang, S., Zhao, Y., Wang, P., & Zhang, J. (2015). *Implementation of training convolutional neural networks*. arXiv preprint arXiv:1506.01195.
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.