Plant External Structure Classification System Based on Convolution Neural Networks

Qing Li
Concordia University
Montreal, Canada
liqing51269@gmail.com

Gaoshuo Cui
Concordia University
Montreal, Canada
cuigeorge918@gmail.com

Jianbin Lai
Concordia University
Montreal, Canada
ca_laijianbin@126.com

Hongran Xu
Concordia University
Montreal, Canada
xuhongran@gmail.com

ABSTRACT

The main purpose of this project is to perform high-precision classification of external structure of plants. It is mainly based on fine-tune VGG-16 network to train the models to extract and learn features of different types of plant structures, so that we can find the appropriate spatial classification boundaries and place different types of samples in their respective regions.After the training is completed, even the unknown new samples can be classified accurately and quickly.

1 INTRODUCTION

In this report, it mainly includes three parts: image preprocessing, model training and result analysis.

For the preprocessing part, we first divide the dataset into five categories, then adjust the size of the image, and solve the problem of uneven category. And then we used VGG-16 with fine-tune as our training model, and combined with K-fold cross validation to further improve the prediction accuracy of the model.

Finally, we clearly show the loss and accuracy of each round of cross validation, as well as the high prediction accuracy after inputting untrained plant images. For example,if we input something like the flower on the left in Figure 5. After the calculation of the model, a set of predicted values (the right side of Figure 5) will be obtained, the classification with the largest predicted value is the final result.

2 PREPROCESSING

The number of training data is a problem that must be considered. The most essential thing to consider is how many samples can effectively cover the problem space, because if the samples are too small to cover the entire problem space, then there will be a lack of information. For example, if we only have one data to train the network, even if the system fits well, it still cannot generalize to the problem space. Therefore, we need to choose a dataset with sufficient representative sample data, meanwhile, the number of samples is also enough. So that we can lay the foundation for good training of the network.

2.1 Dataset Selection

After a lot of search and comparison, we finally selected the dataset on ImageCLEF 2013[2] as our training and testing data. ImageCLEF 2013 are part of the Pl@ntView dataset which contains various pictures of plants on many species of herbs and trees mainly from French area. It contains scans and photographs. Scans are exclusively focussing on one single leaf, while 5 categories(leaf, flower, fruit, stem and entire) of photographs "into the wild" are focusing on different subparts and views of a plant which contains a total 26077 images. All images are in *jpg* format, each image is uniquely identified by an integer "uid" between 1 and 30000. Each image is named by uid.*jpg*. The *xml* file containing the meta-data of an image identified by uid is named by uid.xml.

We plan to use 80% of the entire dataset as training data and the remaining 20% as test data. Then we use 80% of the training data for training and 20% of the data for verification. Equivalent to each round of cross-validation, the train data accounted for 64% of the total dataset, the validation data accounted for 16% of the total data set, the last 20% is the test set.

2.2 Image Scaling

Our dataset contains the *jpg* format and its corresponding *xml* format. The first thing is to separate the files of these two formats, and then classify all datasets into five subcategories(entire, flower, fruit, leaf, stem) according to the tags recorded in the uid.xml. Second, since the size of the image on the dataset is 600*800/800*600, once the image is stored in the form of an array, an image needs to be 351MB, which will cause a lot of time cost for the training model, so we have to appropriately reduce the size of the image.

Image scaling is achieved by reducing the number of pixels. Therefore, it is necessary to select appropriate pixels from the original image according to the reduced size, so that the characteristics of the original image can be maintained after the image is reduced. We choose the Lanczos filter to scale the image, which is used for image scaling and rotation in the field of image processing, and can achieve the best balance in reducing aliasing, sharpness, and minimal ringing. The specific algorithm flow as shown (Figure 1).

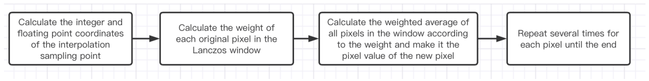

Calculate the integer and floating point coordinates of the interpolation sampling point
Calculate the weight of each original pixel in the Lanczos window
Calculate the weighted average of all pixels in the window according to the weight and make it the pixel value of the new pixel
Repeat several times for each pixel until the end

Figure 1: An illustration of Lanczos filer algorithm.

So, what size images are appropriate? We should try to scale the image as much as possible without losing the image feature information. We tried the three cases of 64*64, 84*84,100*100, and found that after scaling to 64*64 or 84*84, the model does not work well. The reason behind it may be that too much image information is lost. So in the end we decided to reduce the size of the picture to 100*100.

2.3 Penalty Weight of Positive and Negative Samples

After the completion of the above work, an obvious problem appeared. We found that the number of images in these five categories was uneven, of which the leaf category was the most, ten times that of the other categories. The number distribution in the training set is: entire-1455, flower-3522, fruit-1387, leaf-13284, stem-1337. This is not difficult to understand, because for a plant, the number of leaves is generally higher than that of fruits and flowers. Even many types of plants will not bloom or have fruits, such as Algae, mosses and ferns.

Unbalanced sample categories will result in too few features in a classification with a small sample size, and it is difficult to extract rules from it; even if a classification model is obtained, it is easy to produce over-reliance and limited data samples and lead to overfitting problems. When it comes to new data, the accuracy of the model will be poor.

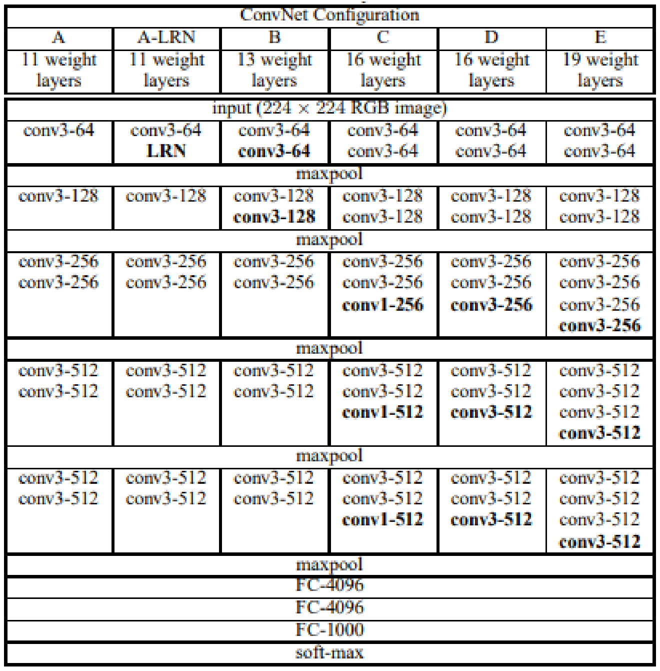
The idea of solving the problem of unbalanced samples by the penalty weight of positive and negative samples is to assign different weights to the categories of different sample numbers in the classification (in general, the weight of small sample size category is high, while the weight of large size sample category is low). The implementation method is to use the Sklearn API: *class_weight*. The specific process is set *class_weight* =’*balanced*’, at this time, *compute_class_weigh* is automatically imported from sklearn.utils. *class_weight* to calculate the weight. In this way, the weight of each category in the input sample can be balanced. The calculation formula is: *Weight* = n_samples /n_classes * np.*bincount(y)*. Where np.*bincount(y)* is the number of samples in each category. After calculation, the weight values of the five categories are shown (Table 1).

Table 1: Penalty Weight of different categories.

Categories	entire	flower	fruit	leaf	stem
Weight	2.8858	1.192	3.026	0.316	3.140

3 TRAINING NETWORK

VGG-16[3] is a popular convolutional neural network model in this field, the specific model of VGG-16 is shown in (Figure 2). In our case, the model combines it with the fine-tuning solution we have implemented. As for the structure of VGG-16 with fine-tune in this project, it is categorised into 5 blocks of convolutional layers and 1 block of fully-connected layers. In addition, it would use K-fold technology to lead a group of more accuracy result. At last, it could use the trained model to predict.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2: An illustration of VGG-16 network[3]

3.1 VGG-16 With Fine-tuned

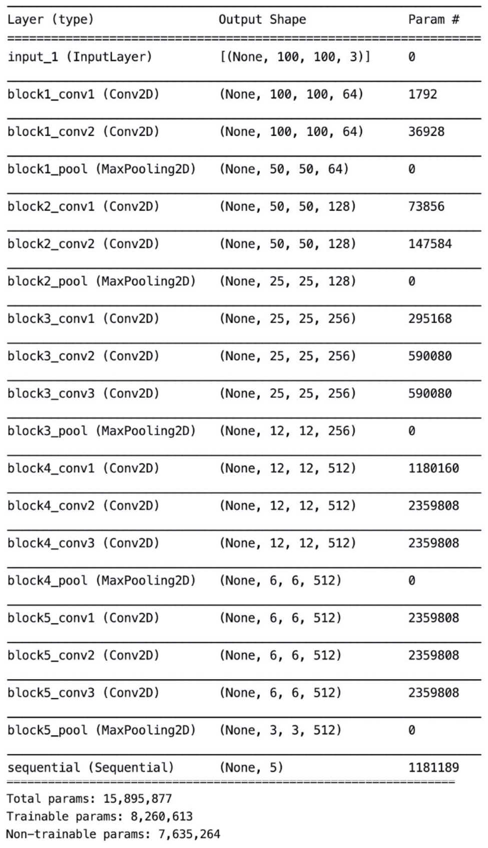
Because in VGG-16, the weight is tuned in a good condition, in this project, it would use parts of weight in VGG-16 as a fine-tune solution[1]. To be specific, the non-top VGG-16 model, which contains 5 blocks of convolutional layers, is loaded as the base model with its weight trained from ImageNet, then it applies one block of two fully connected layers as the top model. The base model and the top model are connected through making the output of the base model be the input of the top model. In this project, we tries to freeze the weight of 14 layers (including maxpooling layer) in VGG-16 and makes the weight in rest layers trainable to adapt to our dataset.

The structure of our fully connected block is implemented by using the *Sequential()* method to group a linear stack of two dense layers along with a dropout layer. Dense layer is used to fully connected the neural of the output of the base model and make non-linear features to be separable.

The first dense layer outputs a shape of (*,256) dimension with ReLU activation function, followed by a Dropout regularization method, this is, some dimensions in the input samples are randomly deleted to avoid overfitting. The parameter in dropout represents the proportion of features to be discarded in each training, which is set to 0.5 in this experiment, that is to say, when training and predicting the model, 50% of the number of features will be deleted each time. Because when half of the hidden layer neurons are cleared, a neuron can’t only rely on the specific neurons associated with it. Dropout forces a neural unit to work with other randomly selected neural units. Elimination weakens the joint adaptability of neuron nodes, enhances the generalization ability, and achieves good results.

The second dense layer outputs a shape of (*,5) dimension corresponding to the 5 classification categories and applies the softmax function. The output size and total parameters of each layer in VGG-16 are shown (Figure 3).

Furthermore, when compiling the model, it uses stochastic gradient descent method optimizer, and to computes the categorical cross entropy loss. Initially the class weight of the imbalanced data set would be used in training which has been computed before(Section 2).



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 100, 3)]     0

block1_conv1 (Conv2D)        (None, 100, 100, 64)      1792

block1_conv2 (Conv2D)        (None, 100, 100, 64)      36928

block1_pool (MaxPooling2D)   (None, 50, 50, 64)        0

block2_conv1 (Conv2D)        (None, 50, 50, 128)       73856

block2_conv2 (Conv2D)        (None, 50, 50, 128)       147584

block2_pool (MaxPooling2D)   (None, 25, 25, 128)       0

block3_conv1 (Conv2D)        (None, 25, 25, 256)       295168

block3_conv2 (Conv2D)        (None, 25, 25, 256)       590080

block3_conv3 (Conv2D)        (None, 25, 25, 256)       590080

block3_pool (MaxPooling2D)   (None, 12, 12, 256)       0

block4_conv1 (Conv2D)        (None, 12, 12, 512)       1180160

block4_conv2 (Conv2D)        (None, 12, 12, 512)       2359808

block4_conv3 (Conv2D)        (None, 12, 12, 512)       2359808

block4_pool (MaxPooling2D)   (None, 6, 6, 512)         0

block5_conv1 (Conv2D)        (None, 6, 6, 512)         2359808

block5_conv2 (Conv2D)        (None, 6, 6, 512)         2359808

block5_conv3 (Conv2D)        (None, 6, 6, 512)         2359808

block5_pool (MaxPooling2D)   (None, 3, 3, 512)         0

sequential (Sequential)      (None, 5)                 1181189
=================================================================
Total params: 15,895,877
Trainable params: 8,260,613
Non-trainable params: 7,635,264
```

Figure 3: Fine-tune with VGG-16

3.2 K-fold Cross Validation

To derive a more accurate estimate of model prediction performance, the program applies k-fold cross validation technology. In this project, the program sets K to be 5, and in each split, the data would be splitted into 5 groups, and then it would take one of the groups to be validation data, in this project, in each split, there are about 16,000 images to be training data, and there are about 4,000 images to be validation data. During the progress, the validation data would also be shuffled, so in each split, the validation would be different, which makes the result more objective.

We use ImageDataGenerator to further expand the data set, strictly speaking, the more data for training, the better the classification effect will be produced. So, let the original image flipped and stretchedwhich not only increase the training set can also prevent overfitting in cross-validation. The picture after ImageDataGenerator processing is shown (Figure 4).

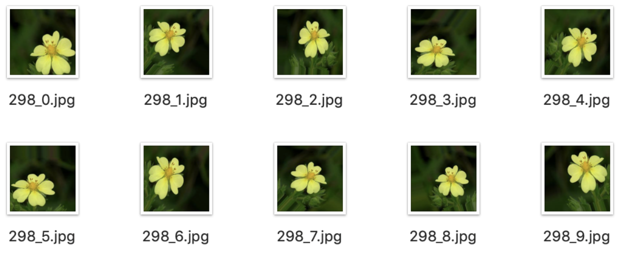


Figure 4: Image flip and stretch

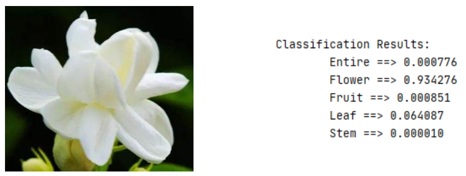
In this project, we have tried different epochs (1/5/10/15/20). For example in 5 epoch conditions, in each split, in each epoch, it will validate after the training step to lead a better estimation. After finishing one of the splits, the final validation loss and validation accuracy would be recorded. Furthermore, it would also record the result of train loss and train accuracy of each epoch in each split.

3.3 Dealing with the result

Finally, as the 5 epoch example mentioned above, it would get 5 groups of these validation loss and validation accuracy, in this case, it computes the mean of the validation accuracy and the standard deviation accuracy. After getting the result, it would save the model into a file called "cnn_model.h5" in the disk, furthermore, it saves the *LabelBinarizer* Entity about the label of this project in the disk. Moreover, it could use the saved record about the result of train loss and train accuracy of each epoch in each split to plot the figure. In this figure, the progress could be shown clearly to the reader (Figure 8, Figure 9, Figure 10, Figure 11, Figure 12). At last, it would use the test data and the model saved before to predict, and then it would show the test accuracy of the data set.

3.4 Predict the Image

Because the model has been saved in the disk, the model could be used to predict an image anytime you want. After predicting the image, it would show a table about the probability of this image in each class (Figure 5), in this figure, it shows that it has more possibility to be flower, which co-respond to the test picture.

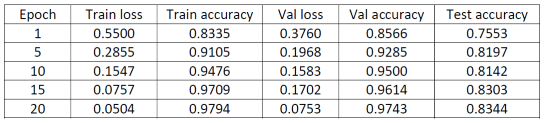


```
Classification Results:
        Entire ==> 0.000776
        Flower ==> 0.934276
        Fruit ==> 0.000851
        Leaf ==> 0.064087
        Stem ==> 0.000010
```

Figure 5: Result from training(5 epoch)

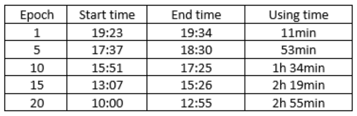
4 TEST RESULT

In this project, the program tries 5 different epoch setting during training data, in each split. The final of each epoch setting result is shown (Figure 6). In this figure, it is obvious to find that, with the increase of epoch, the training loss and validation loss decrease, on the other hand, the training accuracy, validation accuracy and test accuracy climb up. With the increase of epoch setting, the time when training would be more, the using time record in different training condition is shown (Figure 7).



Epoch	Train loss	Train accuracy	Val loss	Val accuracy	Test accuracy
1	0.5500	0.8335	0.3760	0.8566	0.7553
5	0.2855	0.9105	0.1968	0.9285	0.8197
10	0.1547	0.9476	0.1583	0.9500	0.8142
15	0.0757	0.9709	0.1702	0.9614	0.8303
20	0.0504	0.9794	0.0753	0.9743	0.8344

Figure 6: Result difference between different epoch setting



Epoch	Start time	End time	Using time
1	19:23	19:34	11min
5	17:37	18:30	53min
10	15:51	17:25	1h 34min
15	13:07	15:26	2h 19min
20	10:00	12:55	2h 55min

Figure 7: Using time between different epoch setting

The line graph (Figure 8, Figure 9, Figure 10, Figure 11, Figure 12) shows the progress about training loss, training accuracy and validation each time in different epochs setting conditions, it is obvious that the final result when setting epoch to be 20 is the best representation among the cases.

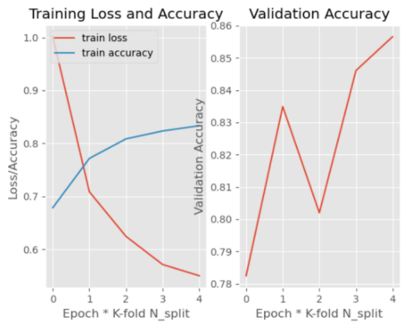


Figure 8: epoch 1

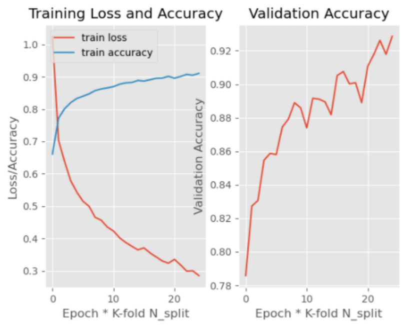


Figure 9: epoch 5

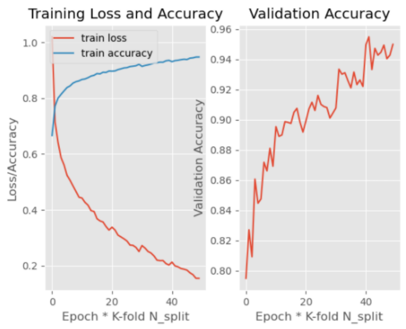


Figure 10: epoch 10

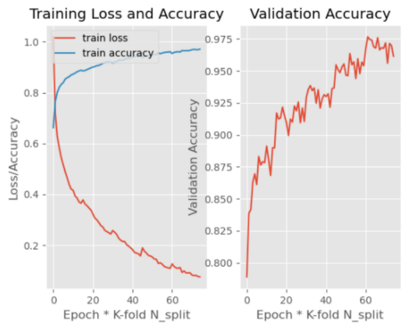


Figure 11: epoch 15

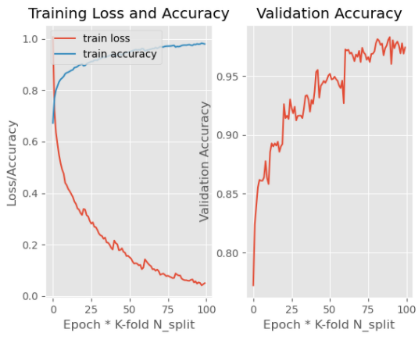


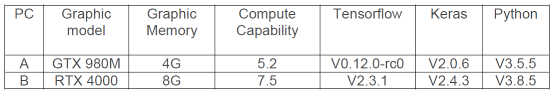
Figure 12: epoch 20

5 CONCLUSION

In this project, our group fine-tuned the VGG-16 model and applied class weights and k-fold cross validation technique on imbalanced dataset, achieving a relatively high accuracy score on the testing dataset. In the experimental result, we could conclude that a small imbalanced dataset could be accurately classified by fine-tuning a convolution neural network model and training model with limited epochs. Moreover, by applying different values(1,5,10,15,20) on parameter 'epoch', we could conclude that the epoch number in a specific range is positively correlated to the accuracy and negatively correlated to the loss. To be specific, the loss of training and validation might drop, along with the increase of the accuracy of training, validation and test if the epoch in the setting is more in a range.

Furthermore, to make this project result more reliable, the program has been applied into different PC settings, and during the process, we found that the representation of the result in PC B condition (Figure 13) is a little bit better than PC A, the factors might be related to the Tensorflow version and Keras version, and graphic model. Some function in old version package might be deprecated, such as the *tf.keras.Model.fit_generator* function has to be changed to *tf.keras.Model.fit*, and the *tf.keras.model.evaluate_generator* function has to be changed to *tf.keras.Model.evaluate* in latest version. To get a good performance of the program, it suggests that programmers should use the latest function.

In the future, beyond this report, the next step of our project aims to implement the logistic regression model on the same dataset and compare the experimental result of them.



PC	Graphic model	Graphic Memory	Compute Capability	Tensorflow	Keras	Python
A	GTX 980M	4G	5.2	V0.12.0-rc0	V2.0.6	V3.5.5
B	RTX 4000	8G	7.5	V2.3.1	V2.4.3	V3.8.5

Figure 13: Different PC settings condition

REFERENCES

[1] Francois Chollet. 2016. Building powerful image classification models using very little data. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html version 0.23.2.

[2] IMAGE CLEF. 2013. Plant Identification 2013. https://www.imageclef.org/2013/plant

[3] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556