Plant Disease Recognition
Using Convolutional Neural Network

Group 9: Deen Huang, Varsha Waingankar, Yupeng Yang

December 08, 2018

Problem & Dataset Description

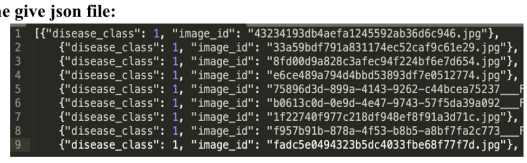
The diagnosis of pests and diseases is essential for agricultural production. We designed algorithms and models to recognize species and diseases in the crop leaves by using Convolutional Neural Network. There is a total of 31,147 images of diseased and healthy plants in our training dataset. These images span 10 original species of plants. Each set of images, including the training, validation, and testing, span 61 different species diseases of plants. Original images have various 2-dimensional sizes with different names.

Exploratory Data Analysis

Data Format Processing: There are 3 image information forms we downloaded from the competition website, including a image names file, a json file with disease class label and image IDs and a label table with label names IDs and names. For future analysis purpose, we had to preprocess these original data forms to an integrated new data frame. After merging these tables by matching their IDs and names in python, we get a new data form consists of plant species, label IDs, diseases names and image names (shown as the left picture below).

Creating a Pandas DataFrame from the give json file:

Image names file



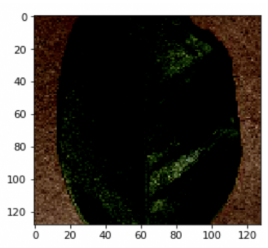
Json file with disease class and image name

Specise	Label ID	Label Name
Apple	0	apple healthy
	1	Apple_Scab general
	2	Apple_Scab serious
	3	Apple Frogeye Spot
	4	Cedar Apple Rust general
Cedar	5	Cedar Apple Rust serious
	6	Cherry healthy
Cherry	7	Cherry_Powdery Mildew general
	8	Cherry_Powdery Mildew serious
Corn	9	Corn healthy
Cercospora	10	Cercospora zeaemaydis Tehon and Daniels general
	11	Cercospora zeaemaydis Tehon and Daniels serious
Puccinia	12	Puccinia polysora general
	13	Puccinia polysora serious
	14	Corn Cercospora leaf spot fungus general

Label table with label id and label name

	species	label_id	label	image_name
0	Citrus	25	Citrus Greening June general	f90a2593f3e845c22641fb18f33d1889.jpg
1	Grape	17	Grape heathy	feaf1075efa9ed6167dec2370bbc2e2e.jpg
2	Tomato	41	Tomato healthy	c40cefa639468c7db5ae6f314ca8c995.jpg
3	Potato	33	Potato healthy	c1759ec6-d90c-468b-9cd6-243e40e5893f___RS_HL 4...
4	Tomato	59	Tomato YLCV Virus serious	ed284bbd1d170bd2816f9b8f8db232a6.jpg

Final dataframe after preprocessing data



	Before	After
Height	429px	128px
Width	256px	128px
RGB range	[0,255]	[-1,1]

Image Resizing & Scaling: Ensure that the images have the same size and aspect ratio (assume a square shape of input image for future modeling). After resizing our images data to square shape, each of our final images becomes 128px height×128px width by our predetermined aspect ratio (shown as the right picture above).

Image color values normalization: Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network. We normalized image data in the RGB range from [0, 255] to [-1,1] (shown as the right picture above).

Models Implemented (VGG Architecture)

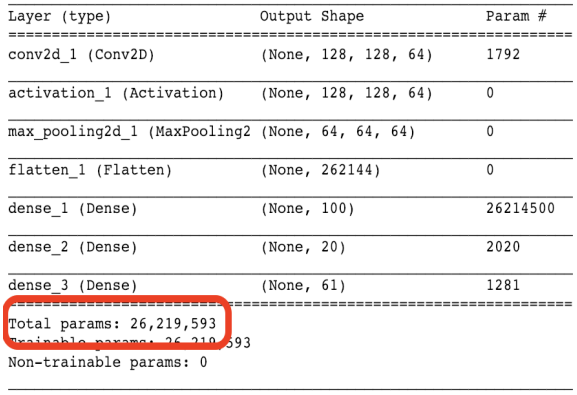
Before the problem of crop disease detection can be solved, the problem of identifying different species of plants need to be addressed. Our work moved towards using convolutional networks to get better performances. Convolutional Networks are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. Convolutional Nets have been successful in identifying faces, objects, and traffic signs apart from powering vision in robots and self driving cars.

ConvNets derive their name from the "convolution" operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small filters. Here are some parameters we used in our ConvNets models:

Depth: Depth corresponds to the number of filters we use for the convolution operation.
Stride: The number of pixels which we slide out our filter matrix over the input matrix.
Activation function: ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation element wise matrix multiplication and addition, so we account for non-linearity by introducing a nonlinear function like ReLU).
Spatial Pooling: (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. These networks have grown in the number of layers leading to architectures such as ResNet and AlexNet that have been trained on images such as Cifar-10 and then fined tune to other problems, such as plant classification.

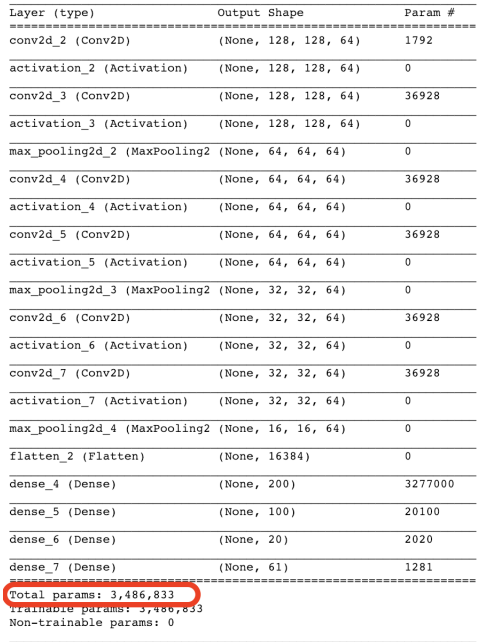
Basic CNN vs. Deeper CNN

Number of Parameters: Two models were implemented. The left picture shows the output of basic CNN total parameters number, while the right picture shows our deeper CNN parameters number. After a convolution layer, it is common to add a pooling layer in between CNN layers. The function of pooling is to continuously reduce the dimensionality to reduce the number of parameters and computation in the network. This shortens the training time and **controls overfitting**. Actually, the **deep cnn yields better accuracy** because, it has more max pooling layers and fewer parameters/features to train on, helps in dimensionality reduction.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 64)	1792
activation_1 (Activation)	(None, 128, 128, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 64)	0
flatten_1 (Flatten)	(None, 262144)	0
dense_1 (Dense)	(None, 100)	26214500
dense_2 (Dense)	(None, 20)	2020
dense_3 (Dense)	(None, 61)	1281

Total params: 26,219,593
Trainable params: 26,219,593
Non-trainable params: 0



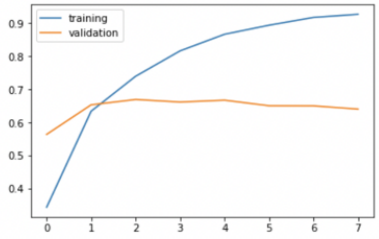
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 128, 128, 64)	1792
activation_2 (Activation)	(None, 128, 128, 64)	0
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928
activation_3 (Activation)	(None, 128, 128, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 64, 64, 64)	0
conv2d_4 (Conv2D)	(None, 64, 64, 64)	36928
activation_4 (Activation)	(None, 64, 64, 64)	0
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36928
activation_5 (Activation)	(None, 64, 64, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 32, 32, 64)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36928
activation_6 (Activation)	(None, 32, 32, 64)	0
conv2d_7 (Conv2D)	(None, 32, 32, 64)	36928
activation_7 (Activation)	(None, 32, 32, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 16, 16, 64)	0
flatten_2 (Flatten)	(None, 16384)	0
dense_4 (Dense)	(None, 200)	3277000
dense_5 (Dense)	(None, 100)	20100
dense_6 (Dense)	(None, 20)	2020
dense_7 (Dense)	(None, 61)	1281

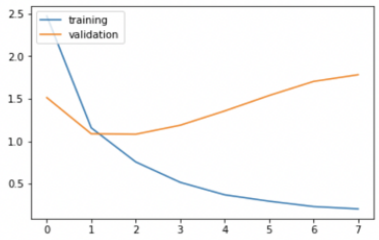
Total params: 3,486,833
Trainable params: 3,486,833
Non-trainable params: 0

Belows are our model implementations outputs, which show the accuracy and loss trends for both of basic CNN model and deeper CNN model.

Implementation 1: Accuracy vs Loss computation and plots for Basic CNN model

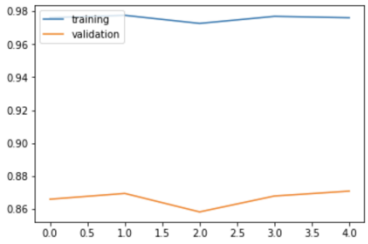
```
Train on 25691 samples, validate on 2855 samples
Epoch 1/8
 - 89s - loss: 2.4712 - acc: 0.3444 - val_loss: 1.5127 - val_acc: 0.5643
Epoch 2/8
 - 88s - loss: 1.1571 - acc: 0.6342 - val_loss: 1.0874 - val_acc: 0.6536
Epoch 3/8
 - 88s - loss: 0.7565 - acc: 0.7398 - val_loss: 1.0823 - val_acc: 0.6697
Epoch 4/8
 - 88s - loss: 0.5168 - acc: 0.8166 - val_loss: 1.1879 - val_acc: 0.6620
Epoch 5/8
 - 88s - loss: 0.3701 - acc: 0.8668 - val_loss: 1.3558 - val_acc: 0.6676
Epoch 6/8
 - 88s - loss: 0.2948 - acc: 0.8941 - val_loss: 1.5374 - val_acc: 0.6508
Epoch 7/8
 - 88s - loss: 0.2321 - acc: 0.9176 - val_loss: 1.7031 - val_acc: 0.6504
Epoch 8/8
 - 88s - loss: 0.2046 - acc: 0.9270 - val_loss: 1.7817 - val_acc: 0.6406
CPU times: user 7min 52s, sys: 2min 49s, total: 10min 41s
Wall time: 11min 45s
```

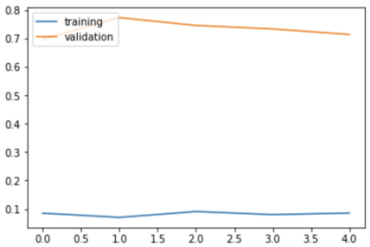




Implementation 2: Accuracy vs loss computation and plot of Deeper CNN model

```
Train on 22836 samples, validate on 5710 samples
Epoch 1/5
 - 78s - loss: 0.0849 - acc: 0.9761 - val_loss: 0.6984 - val_acc: 0.8660
Epoch 2/5
 - 77s - loss: 0.0702 - acc: 0.9775 - val_loss: 0.7740 - val_acc: 0.8695
Epoch 3/5
 - 77s - loss: 0.0907 - acc: 0.9726 - val_loss: 0.7465 - val_acc: 0.8583
Epoch 4/5
 - 77s - loss: 0.0798 - acc: 0.9769 - val_loss: 0.7342 - val_acc: 0.8680
Epoch 5/5
 - 77s - loss: 0.0854 - acc: 0.9760 - val_loss: 0.7148 - val_acc: 0.8709
CPU times: user 3min 53s, sys: 1min 16s, total: 5min 9s
Wall time: 6min 26s
```

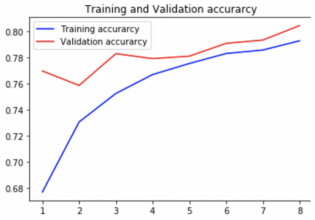




Note: The results in implementation 2 was received by attempting several times of changing epochs.

Implementation 3: Accuracy vs Loss of Deeper CNN with Image augmentation

```
Epoch 1/8
1784/1784 [==============================] - 103s 58ms/step - loss: 0.9823 - acc: 0.6770 - val_loss: 0.6676 - val_ac
c: 0.7695
Epoch 2/8
1784/1784 [==============================] - 103s 58ms/step - loss: 0.7629 - acc: 0.7307 - val_loss: 0.7092 - val_ac
c: 0.7585
Epoch 3/8
1784/1784 [==============================] - 102s 57ms/step - loss: 0.6927 - acc: 0.7522 - val_loss: 0.6129 - val_ac
c: 0.7828
Epoch 4/8
1784/1784 [==============================] - 102s 57ms/step - loss: 0.6457 - acc: 0.7669 - val_loss: 0.6473 - val_ac
c: 0.7790
Epoch 5/8
1784/1784 [==============================] - 102s 57ms/step - loss: 0.6161 - acc: 0.7754 - val_loss: 0.6031 - val_ac
c: 0.7809
Epoch 6/8
1784/1784 [==============================] - 103s 58ms/step - loss: 0.5842 - acc: 0.7825 - val_loss: 0.5592 - val_ac
c: 0.7907
Epoch 7/8
1784/1784 [==============================] - 103s 58ms/step - loss: 0.5684 - acc: 0.7857 - val_loss: 0.5872 - val_ac
c: 0.7932
Epoch 8/8
1784/1784 [==============================] - 103s 58ms/step - loss: 0.5523 - acc: 0.7925 - val_loss: 0.5727 - val_ac
c: 0.8042
```



Conclusion:

After implementations of 1 Basic Convolutional Neural Network models and 2 Deep CNN models, we increased our validation accuracy by image augmentations but decrease the training accuracy as cost. Regarding to the challenge of low accuracy we met in prediction of basic CNN model, we improved it through rescaling the validation data set splitting ratio. In order to promoting the validation accuracy, we tried to “fine-tune” the last convolutional block of the VGG model, which helped to prevent overfitting and increase to a high accuracy as well.

References:

Balaji, Nikhil. “Image Data Pre-Processing for Neural Networks.” Becoming Human: Artificial Intelligence Magazine, Becoming Human, Sep 11, 2017 becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258

Francois Chollet, “Building powerful image classification models using very little data”, June 05, 2016 https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition", Apr 10, 2015 https://arxiv.org/abs/1409.1556