Xiaodan Chen, Hyunjae Yu
DATS 6203
Professor Chen Zeng
Final Project Report

Plant Diseases Classification

Introduction

Plant diseases are major threats for smallholder farmers, who depend on healthy crops to survive, and about 80% of the agricultural production in the developing world is generated by them. Identifying a disease correctly when it first appears is a crucial step for efficient disease management and agricultural production. Our objective of this project is to design algorithms and models to recognize species and disease in the crop leaves.

We downloaded our dataset from the AI Challenger Competition (https://challenger.ai/competition/pdr2018). The dataset was randomly splitted into two sub-data sets: training (31784 images), testing (4540 images). Each image contains one leaf occupying main position of the image.The training set contains the json file of the image and the annotation, and the json file contains each image and the corresponding category ID.

There are a lot of models we can choose since the release of LeNet5 to deal with the image recognition issues. Later on there are AlexNet, VGG, Google Inception, ResNet, DenseNet neural network structures for CNN issues. For this project, we just use the revised baseline model and Densenet model, which is released in 2017, to deal with this issue. The latter network used in our project are inspired by the article "Densely Connected Convolutional Networks" and open source implementation. The network and its variants can achieve a very high accuracy among different public image datasets according to this article. In spite of its theoretical accuracy in classification, there are some drawbacks to use DenseNet. DenseNet and its variants mentioned in the article may not feasible for our project due to the computation time limitation. Moreover, they were evaluated with a completely different environment. However, we still believe that it is still a good learning opportunity to just apply such networks to our project.

Methodology

- *Baseline Model on AWS*

The baseline model is actually a multiple-layer neural network. For the data cleaning part, we resize all training images to the size of 128 to meet the need of the model. For the modeling part, we try several combinations of the network and there is no big difference as with the result of the accuracy. Finally we choose one convolution layer, one max-pooling layer, one fully connected layer as our network structure, feeding the input data with every batch size of 32 to fit. We also use the drop-out method to get rid of overfitting. For the evaluation part, we take cross entropy as the method of loss function. As the loss goes down, the final accuracy keeps around 35.4% after 2500 steps.

Layer	**Convolution**	**Max-Pooling**	**Fully Connected**
Parameters	5*5*3*64 + 64	3*3*64/2	(64^3)*61

- *Densenet*

DenseNet is basically a chain of dense blocks, transition layers between each block and one output layer at the end. DenseNet uses a residual network style structure, which is named as dense block according to the article. In every dense block, each layer will have direct connections to all subsequent layers. Each dense block chains up several concatenations. By concatenating the input and normalized convolution together as the output, DenseNet adds shortcuts among layers during the forward pass. Hence the backward pass is very much like a residual add.

We constructed a DenseNet-201 model by using the torchvision package;the torch models are trained under the same setting. The table below shows the architecture of the pretrained models.

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

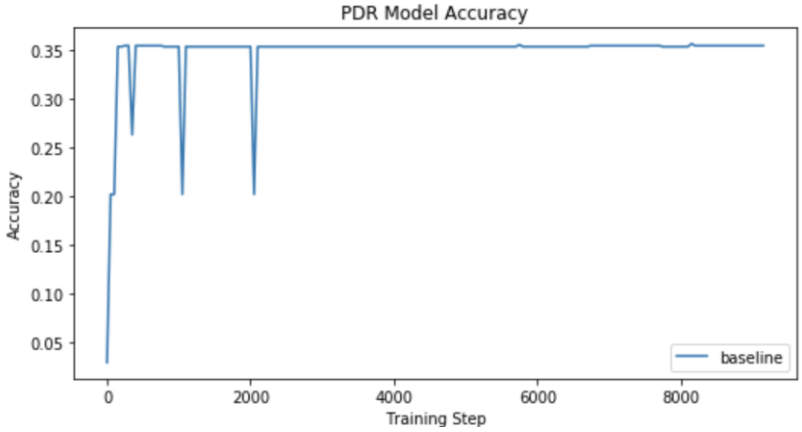
Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Since all pre-trained models expect input images normalized input images in the same way, the images are loaded into a range of [0,1] and then transformed into a form of mini-batches of 3 channel RGB images of shape (3*H*W), where H and W are expected to be at least 224. When feature extracting, we want to compute gradients for the newly initialized layers so we can have all of the other parameters do not require gradients. Therefore, we created a helper function that sets the '.requires_grad' attribute of the parameters to True and the parameters for the layer we are reshaping. Then, we define a loss function and optimizer by using a Classification Cross-Entropy and SGD with momentum. Lastly, we train the network; we create a loop over our data iterator and feed the inputs to the network and optimize.

Results

- *Baseline Model*

From baseline model, although its 35.4% accuracy is not high, we do have a deeper understanding of the limitation of shallow network. In order to run this model more quickly and easily, we deploy this model on AWS. From this process we learn a lot of configuration methods and the importance of setting parameters appropriately of an deep learning instance.



- *Densenet*

After we train and evaluate our model using DenseNet, we achieved 86% accuracy on test set with a cross entropy loss of 1.05.

```
[Epoch 0] train loss 1.057316 train acc 0.867268  valid loss 1.020904 valid acc 0.874890
.....
[Epoch 1] train loss 1.058728 train acc 0.865849  valid loss 1.021330 valid acc 0.874449
.....
[Epoch 2] train loss 1.061733 train acc 0.864903  valid loss 1.020121 valid acc 0.874890
.....
[Epoch 3] train loss 1.061305 train acc 0.865660  valid loss 1.019529 valid acc 0.875771
save model...
saved.
.....
[Epoch 4] train loss 1.056393 train acc 0.865944  valid loss 1.022011 valid acc 0.875110
.....
[Epoch 5] train loss 1.054995 train acc 0.867615  valid loss 1.020799 valid acc 0.874229
.....
[Epoch 6] train loss 1.056885 train acc 0.866227  valid loss 1.021436 valid acc 0.875110
.....
[Epoch 7] train loss 1.058135 train acc 0.865218  valid loss 1.020178 valid acc 0.874229
.....
[Epoch 8] train loss 1.056503 train acc 0.867425  valid loss 1.020390 valid acc 0.874009
.....
[Epoch 9] train loss 1.058139 train acc 0.865218  valid loss 1.020801 valid acc 0.873568
```

- *Comparison*

Model	**Baseline Model**	**DenseNet**
Accuracy	35%	86%

Conclusion

From the project, we have seen that some of the DenseNet variants out-perform the baseline network. However, we realize that there are some negative factors, such as imbalanced data. In addition, time was the main problem. DenseNet has an obviously more complex structure than the baseline network and thus will take more time to train per epoch. We thought a fewer number of trainable parameter in the network might lead to less time on training. However, it seems that the structure of the network is also a factor of training. Thereby we trained and evaluated the models with 10 epochs. As a consequence, DenseNet may not be fully trained. Moreover, we used the default values for mean, std, RandomResizedCrop, and RandomRotation which are the values used in ImageNet. If we calculate the actual mean and the standard deviation, it may have improved the result better. In spite of all the negative factors on our project, DenseNet still shows some positive aspect from a perspective of its potential. Though our DenseNet model was not fully trained, it achieved a better score than the baseline model.

Reference

https://arxiv.org/abs/1608.06993

https://github.com/liuzhuang13/DenseNet

https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html

https://pytorch.org/docs/stable/torchvision/models.html

https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

https://pytorch.org/tutorials/advanced/neural_style_tutorial.html

https://pytorch.org/tutorials/beginner/saving_loading_models.html