Cassava Leaf Disease Classification

Asher Hanson
Fall 2021 CS 474-41: Deep Learning, Dr. Xian
University of Idaho
Moscow, Idaho
asher.2018@outlook.com

I. Introduction

In Sub-Saharan Africa, there is a plant called cassava. This plant in recent years, has become threated by many diseases which in turn, are affecting the harvest yield. The cassava plant is one of the main providers of carbohydrates and is grown by nearly 80% of farmers in the region. One problem that currently resides is the lack of knowledge concerning how to correctly identify the disease that compromises the growth on a cassava. As of right now, the main solution to this problem has been reaching out to a government official to come inspect the plant and offer a diagnosis and treatment. This solution has become labor-intensive and costly, which has led community members to ask is there a better solution.

As a possible solution, members of the National Crops Resources Research Institute have collected and identified over 21,000 images of the cassava plant with different visible illnesses. The goal has become to create a model that has the best accuracy in identifying diseases based off an image. The hope is by doing so, farmers will be able to take pictures of their plants and get quick responses to how to help their crop without too much time passing or spending much money.

This paper will go over the steps done to create a Convolutional Neural Network that has the capacity to both receive an image of a cassava plant and then predict the imposing diseases.

II. Proposed Methods

A. *Model Architecture*

To begin this project, images were imported from the training images given by the National Crops Resources Research Institute and then passed through a TensorFlow image processor. The purpose of the processor is to separate the receiving images into training and validation, along with stating the image size, batch size, and seed.

The complete model from this project consists of two parts, the first part entails a model that was downloaded/installed, and the other, layers that were added manually. In this section, the model and layers added will be named, but the specific parameters will be provided and explained in a future section.

The model imported was VGG16; this model is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford. The model consists of several layers with different functions. In this model, there are five main blocks. Each block is played out in the following design:

1. Convolution + RELU
2. Convolution + RELU
3. Max Pooling

The reason VGG16 was implemented was because of its reputation in working with image classification. One of VGG16's benefits is that instead of having many hyper-parameters, it focuses on having convolution layers of 3x3 filter with a stride 1 and always uses same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture.

Following the five blocks shown above, there are the handwritten layers, these layers are called the Fully Connected Layers and are:

1. Flatten Layer
2. Dense + RELU
3. Dropout
4. Dense + RELU
5. Dense + RELU
6. Dense + SOFTMAX

The lines of code after the VGG16 were added because they were increased the accuracy and validation accuracy. The final accuracies and performances will be discussed later, the main purpose of this section is to highlight the different layers in the architecture.

B. Parameter Setting

In this section, we will elaborate on parameters that were assigned to each layer along with those assigned when fitting the model.

To begin, the first thing decided was the image size. More pixels mean more specific/content, but it also means more time and training. This model was built with the image size 256x256.

In the order of sequence, after resizing the images and splitting them into sections, the images were passed through the VGG16. The weights were set to 'imagent', meaning these weights were trained on Imagenet dataset.

Next, came the decision to exclude the initial layers from training phase as they are already trained.

After setting the VGG16 parameters, came the parameters of the Fully Connected Layers. The specific units and activations will be listed in Table1.

When compiling the model, several important parameters were set. First, the loss was set to be categorical crossentropy. The reason this was decided was because the images are categorical and contain more than one classification. Next, came the optimizer. For this model, Adam was decided, this was because of 'Adams' ability to create a gradual loss for each epoch. Something that helped this model was setting a learning rate, this model has it set to (1e-6).

The final parameters set were in the fitting of the model. Here, steps-per-epoch were set to 170, epochs to 42, and validation steps to 70. It is helpful to have both a large steps-per-epoch and epoch number because these number represent the number of rounds and times your model gets to train. The last parameter set was in the callbacks. Early stopping was used in this model, and it proved extremely helpful. Early stopping monitored the validation loss and would end an epoch early if it noticed that the validation loss was going to be much worse than the previous. It took serval run throughs to find a good patience level, but eventually 20 was decided.

Each parameter selected was done because of its great potential in helping create the most accurate model. The following table summarizes each layer with the set number of parameters.

TABLE I.

Layers	**Model Layers and Parameters**	
	Layer Name	***Parameters***
1	VGG16	14,714,688
2	Flatten	0
3	Dense(Input=4096, activation = relu)	134,221,824
4	Dropout(0.5)	0
5	Dense(Input=4096, activation = relu)	16,781,312
6	Dense(Input=8, activation = relu)	3,277,60
7	Dense(Input=5, activation = softmax)	405

The total parameters count is shown in the following table (Table II). The more trainable parameters there were, the better the model became.

TABLE II.

Total Parameters	
Total Params	166,045,989
Trainiable Params	151,331,301
Non-trainable Params	14,714,688

Up to this point in the paper, an explanation for the architecture and the parameters have been given, up next will be the diagnosis of how the model performed.

III. EXPERIMENTAL RESULT

After running combining, compiling, fitting, and running the model, accuracies from the training and validation data were gathered. The following table shows:

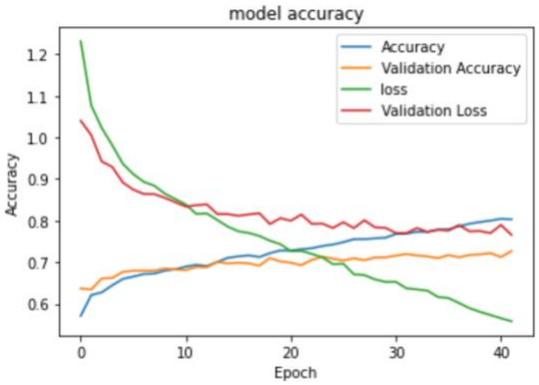
A. Training Performance

TABLE III.

Data	**Loss**	**Accuracy**
Training	0.5577	0.8029
Validation	0.7656	0.7226

Once the model was done running, the following plot was made to demonstrate how the loss and accuracy changed over the course of the 42 epochs.

FIGURE 1.



From the plot, there are a few fascinating traits worth explaining. First, we can tell that the current learning rate is doing well because of the lack of hills in the lines. Next, it is important to note that the loss curves, both the loss and validation loss, were both going down steadily. We also observe that the Accuracy and Validation Accuracy both had a positive slope. From this plot, I predict that if the model would have been running for more epochs, a higher accuracy and validation accuracy could have been achieved.

B. Test Performance

Once the model was completed, test data, (data given by the professor), was rather through the model and predicted. From the test data, two things were collected, one, the value count of how times each label was used, and two, the test accuracy.

TABLE IV.

Data	Accuracy
Test	0.711

TABLE V.

Label	Value Count
4	473
3	3070
2	214
1	407
0	116

An important observation about this model is that the validation accuracy is a good predictor of what the test accuracy will be. The validation accuracy was 0.72 and the Test Accuracy was 0.71. My guess is that if the model would run longer, both the validation accuracy and test accuracy would increase.

IV. CONCLUSION

A. Problems Observed

When building models, there will be a few problems that arise, in this section, a few of those problems will be discussed.

1. **Time:** While creating models, it took about two and a half hours to run. So, if we wanted to run the model five times, making small adjustments to the model after each run, it would take close to twelve and a half hours.
2. **Parameters:** There are so many different parameters to use, and sometimes, you don't know which ones to use or what to set them too. A lot of model building is trial and error.
3. **Model Architecture:** It is a good idea to do research on different model structures before trying to build new ones. Many times, there are models already created that look exactly how you would like to build one. VGG16 proved to be very good in this model, but there might be other models more helpful.

B. Possible Outcomes

1. **Epochs**: I would like to run more model for 60-70 epochs and see how my model performs. According, to my plot, it looks like the validation accuracy was starting to level out, but the losses were still decreasing. So, I think my model could be improved in that sense.
2. **Parameters**: When I began making my model, I used a callback call ModelCheckpoint that also helped monitor the val_loss. I would like to go back and see if that parameter could help.