Recognizing Diseased Coffee Leaves Using Deep Learning CS230-Fall 2020

Shelly Deng
Department of Computer Science
Stanford University
jsdeng@stanford.edu

Abstract

Since coffee is widely consumed, we aim to build a coffee plant disease recognition algorithm, which classifies coffee leaf images into one of six health conditions: Healthy, four levels of Rust, and Red Spider Mite infestation. We first tested a variety of neural networks. Our fully connected neural network (FCNN) and an AlexNet-inspired convolution neural network (CNN) achieves only 40.3% and 59.4% accuracy respectively, but our ResNet18 model with transfer learning achieved accuracy of 75.3% on the test set. We then attempted a two-task approach by 1) separating the leaves based on the three types of health conditions and 2) separating Rust leaves based on severity using a regression-based classification. With regularization, dropout, and weighed sampling, we achieved 86.4% and 82.6% accuracy for the three-class and regression-based classification tasks respectively. However, the macro averaged F1 scores are still low due to insufficient data and class imbalance.

1 Introduction

Coffee is the second most widely consumed beverage in the world after water, and it contributes significantly to the world economy. Rust (caused by the fungus Hemileia vastatrix) and pest infestation by Red Spider Mites are two common coffee plant problems. Both, visibly recognizable in the plant leafs, can cause early defoliation, which negatively impacts coffee quality and yield[2]. Early disease recognition and detection are therefore important for quality control in coffee production. In this project, we aim to train deep learning algorithms which can be used for automated recognition of diseased coffee plants based on pictures of coffee plant leaves. Specifically, given a set of images of coffee leaves, this project will output the correct labels for the conditions of the coffee leaves using 1) FCNN, 2) various CNNs with and without transfer learning, and 3) CNNs with a custom-built regression component. We will also iterate on the various deep learning algorithms using different hyperparameters in order find the model that best perform this multinomial classification task of identifying the various conditions of coffee plant leaves.

2 Related work

Researchers have used deep CNNs for a variety of object detection tasks, such as for pedestrian detection[8]. In the similar space, other tasks include guarana plant classification[13], coffee leaf disease recognition[13], tomato plant diseases and pest recognition[2], and coffee grading and disease identification[14]. AlexNet[4,5] with its convolutional layers beat state-of-the-art machine learning model when it was first developed. To attempt to achieve better performance, researcher began implementing deeper networks, which grew in depth and complexity[3]. The introduction of residual networks was therefore revolutionary since it provided a way to reduce complexity of very deep neural network by allowing it to learn identity mapping from earlier to deeper layers[3]. With the prevalence of the use of mobile device, MobileNet and MobileNetV2[12] were CNNs developed with efficiency and performance both in mind.

3 Dataset and Features

3. 1 Composition

For this project, we will be using the robusta coffee leaf images dataset (RoCoLe[9]), which contains 1560 images of coffee leaves total, with 791 healthy leaves and 769 unhealthy leaves. The images were taken by a 5MP, f/2.2 autofocus smartphone camera, and has varying resolution but no smaller than 720x1280. Examples of the images can be seen in Figure 1 in Appendix Section 8.1. We use the following representations: RL = Rust Level, RSM = Red Spider Mite.

Labels	Healthy	RL 1	RL 2	RL 3	RL 4	RSM
Classification Number	0	1	2	3	4	5
Number of Pictures	791	344	166	62	30	167

Table 1: Dataset Composition

3.2 Dataset Preprocessing and Splitting

Since the dataset contains images of varying dimensions, we first resized all the images to be 720x720 (See Figure 1). We then further transform the images by performing center cropping (448x448), add a constant padding of 10, and finally resizing the images to be either 224x224(x3 channels) for all the CNNs and 64x64(x3 channels) for the FCNN. The dataset is splitted the following way: 10% test, 18% validation, and 72% training. Since the dataset contains imbalanced classes, care was taken to ensure that each split contain the right proportion of each of the six classes.

4 Methods

4.1 Two Separate Designs for this Multiclass Classification Task

4.1.1 One-Through Design

The first design involves feeding the full training set containing all six classes into the following neural networks. The details of each of the architectures can be found in the Appendix. Each of the following models were ran for 30 epochs, with learning rate of 1e-4, and batch size of 32.

4.1.1.1 (Fully Connected Neural Network (FCNN)

The fully connected neural network, adopted from the CS230 PyTorch Vision example[7], aimed to set a baseline for this classification task. This dense neural networks consists of three hidden layers, with 3000, 1000, and 200 hidden units respectively. Each of the layers uses a ReLu activation function except for the last, which uses a Softmax activation function.

4.1.1.2 AlexNet

AlexNet[4,5], which is based on LeNet[6], has shown promising results in variety of image classification problems, including pedestrian detection[8], guarana plant classification[13], and coffee leaf disease recognition[13]. AlexNet consists of five convolutional layers followed by 3 fully connected layer, and with ReLu used as the activation function for each layer. The five convolutional layers uses 11x11, 5x5, 3x3, 3x3, 3x3 filters respectively. Max-pooling of 3x3 filters are applied to the first, second, and fifth convolutional layers. Dropout rate of 0.5 is applied to the first two fully connected layers.

4.1.1.3 AlexNet-Inspired CNN

Inspired by the original AlexNet as well as the modified adoptation[2] of the AlexNet used by Sorte et al. in their coffee leaf disease recognition task, we implemented a variation. The customized CNN consists of 4 convolutional layers followed by 2 fully connected layers, and with ReLu used as the activation function for each layer except for the last, which uses SoftMax. The 4 convolutional layers uses 11x11, 5x5, 3x3, 3x3 filters respectively, and max-pooling of 2x2 filters and batch normalization are applied to each of the convolutional layers. Dropout rate of 0.2 is applied to the second fully connected layers.

4.1.1.4 ResNet18[3]

The AlexNet-Inspired CNN and AlexNet have only 6 and 8 layers total. Deeper neural networks have shown to perform well on classification tasks[6], but they risk running into the problem of exploding or vanishing gradient. This makes residual network[3] particularly helpful because it allows for skipped connections between layers. Specifically, the activation of one layer is fed into another layer deeper in the network. Suppose the following residual block where $a^{[\ell]}$ has a shortcut to $a^{[\ell+2]}$ and g is an activation function:

$$a^{[\ell+2]} = g(z^{[\ell+2]} + a^{[\ell]}) = g(W^{[\ell+2]}a^{[\ell+1]} + b^{[\ell+2]} + a^{[\ell]})$$

If $W^{[\ell+2]} = b^{[\ell+2]} = 0$, then $a^{[\ell+2]} = a^{[\ell+2]}$. In other words, this residual block allows the model to learn an identity mapping from the earlier activation $a^{[\ell]}$ to a deeper activation $a^{[\ell+2]}$.

4.1.1.5 MobileNetV2

As farmers might be interested in a model that requires low computational power that can run on mobile devices, we decided to test out MobileNetV2. MobileNetV2 consists of two types blocks: 1) a residual block with a stride of 1 and 2) a block with stride of 2 for downsizing. Each block consists of 3 layers: 1x1 filter with ReLU6, 3x3 depthwise filter with ReLU6, and 1x1 convolution without an activation function. These blocks are repeated a different number of times to construct the full network of MobileNetV2. Detailed architecture can be found in the Appendix.

4.1.1.6 Transfer Learning for AlexNet, ResNet18, MobileNetV2

PyTorchVision[10]'s AlexNet, ResNet18, and MobileNetV2 all have output feature size of 1000. For transfer learning, we apply the following fully connected layers:

Layers	Size	Params	Activation
Input	1000	-	-
Layer 1	128	+ 1000x128 + 128	ReLu
Output	6	128x6 + 6	Softmax

Table 2: Transfer Learning Architecture

4.2 Two-Task Design: With Adoptation of Regression-Based Classification

We decided to also separate this multiclass classification task into a two-step classification problem due to the similarities between the diseased leaves. In this two-step approach, we will first classify all the leaves to be Healthy, RL, or RSM. This will use the same models and transfer learning as above, except as a 3-class classification problem, with output layer having 3 units instead.

In the second task, we use only the healthy and rust-inflicted leaf images and aim to classify each into the correct rust level (0 for Healthy and 1-4 for each of the rust level classifications). We adopted ResNet18 and MobileNetV2 by adding 2 new layers that use ReLu as the activation function and which have 128 and 36 hidden units respectively. Since it maps to a single real-valued output, the real-valued numbers are rounded to the closest integer to represent the prediction of the class (0-5).

4.3 Loss Functions

For each of the model from Design 1 and 2(i), we used the negative log likelihood (NLL Loss) function

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=0}^{k-1} 1\{y^{(i)} = j\} \log(\hat{y}_j^{(i)}),$$

where $k = 6$ for the six-class classification task and $k = 3$ for three-class classification task, n is the number of training examples, and $y^{(i)}, \hat{y}_j^{(i)} \in \{0, ..., k-1\}$. The SoftMax activation layer outputs $\hat{y}^{(i)}$, a vector of probability for each of the classes, and this loss function thus attempts to increase the probability for the correct class.

For Design 2(ii) with the regression-based classification task, we use the mean square error (MSE) loss

$$\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - W^{\top}x^{(i)})^2$$

This attempts to make each prediction $W^{\top}x^{(i)}$ as close to each true label $^{(i)}$ as possible.

5 Experiments, Results, and Discussion

5.1 Testing the Problem Design Frameworks

As a baseline, each of the above neural networks were trained on the full train set (with all six classes) with a learning rate of 1e-4, batch size of 32, and 30 epochs. For metrics, we have calculated the loss and overall accuracy (measured by the percentage of correctly labeled images). Precision ($\frac{TP}{TP+FP}$), recall ($\frac{TP}{TP+FN}$), and F1 ($\frac{precision*recall}{precision+recall}$) scores are calculated for each of the classes using scikit-learn[11] and averaged to get the macro averages.

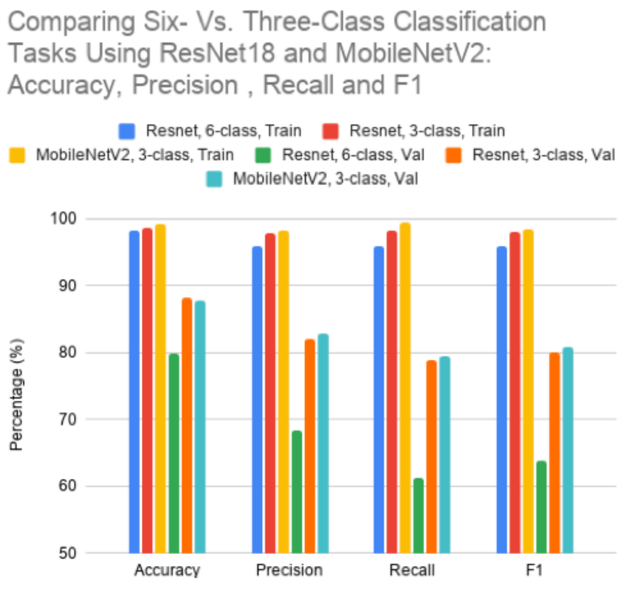
Table 2 shows that the FCNN did not do well on either the training or validation set. We had to reduce the resolution to 64x64x3 to make the input feature size manageable, but low resolution of images makes it harder to detect small diseased spots on the coffee leaves, and without enough hidden layers and units, the algorithm cannot learn the classification task well. The AlexNet-Inspired CNN, AlexNet, and ResNet18 all fit the training set decently with high accuracy and macro

	Train					Val				
Neural Network	**Loss**	**Acc**	**Prec**	**Rec**	**F1**	**Loss**	**Acc**	**Prec**	**Rec**	**F1**
FCNN	0.876	69.8	62.4	64.8	63.1	1.593	40.3	34.2	33.4	32.9
AlexNet-Inspired CNN	8.51e-2	98.1	95.3	96.3	95.7	1.518	59.4	47.9	42.8	44.0
AlexNet	3.76e-2	98.7	96.0	96.8	96.4	1.105	73.4	60.9	59.9	60.0
ResNet18	4.24e-2	98.3	95.9	95.9	95.9	0.939	79.9	68.4	61.3	63.9
MobileNetV2	0.360	88.8	78.8	77.9	77.9	0.729	76.7	58.4	53.6	55.6

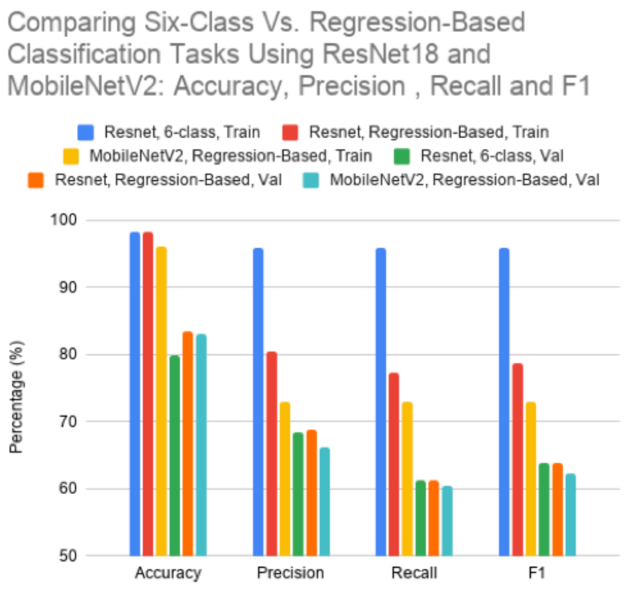
Table 3: Metrics: Loss, Accuracy (%), and Macro Averages (%) for the Six-Class Classification Task

F1 scores. MobileNetV2 has higher loss and lower accuracy and F1 scores comparably, indicating it might require longer training. None of the models generalizes to the validation, suggesting that they've overfitted to the training set.

For ResNet18, the F1 scores for classes 0-5 are 95.1%, 73.6%, 56.7%, 21.1%, 75.0%, and 62.1%. Possible reasons include class imbalance and having insufficient images for classes 2-5. Also, these diseased leaves are hard to classify manually (especially between the rust levels). We thus transformed this task into a two-task problem: regression-based classification might be better suited for classifying the degree of rust. Here, we used only the transfer learning models.



(a)



(b)

Figure 1

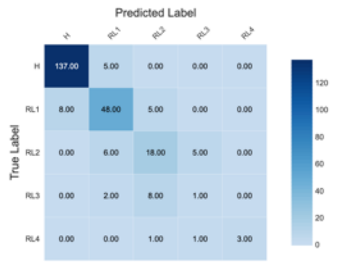
From Figure 1 (a), we see that ResNet18 and MobileNetV2 generalizes to the validation set much better in the 3-class classification task than the best performing model ResNet18 for the 6-class classification task. We can thus verify our previous thought that inability classifying between rust levels contribute to high errors. Figure 1 (b) further confirms that separating rust levels is hard, as seen with the low validation scores for accuracy and macro average precision, recall, and F1. Finally, neither ResNet18 nor MobileNetV2 generalize well in the regression-based task. This might be due to the fact that we simply do not have enough training data for these classes. (Rust Level 2, 3, and 4 only have 121, 45, 22 images in the training set whereas there are 570 images for the Healthy class.)

5.2 Tuning the Best Model

We tried tuning the models with different hyperparameters and mitigations. First, for the insufficient data problem, we've consider random cropping of the center crop to produce more data variation of the same image, but we decided against it. The diseased spots are sometimes small and can exist anywhere on the leaf. This means that random cropping of diseased leaves will likely produce many healthy looking samples, which can increase false negatives.

For the class imbalance problem, we tried sampling the training set inversely proportional to the percentage of the each class, such that the each batch contains roughly the same number of samples from each class. By using PyTorch's BatchSampler and WeightedRandomSampler, we are able to oversample the underrepresented class[1].

According to Figure 2a, we mostly misclassify an image with a neighboring label, again showing that separating between classes is difficult. Figure 2b shows that F1 scores are typically higher for classes with more data. Since validation and



Class	F1 (%)	Support
0	95.5	142
1	78.7	61
2	59.0	29
3	11.1	11
4	75.0	5

(b) F1 and the Number of Images Per Class

Figure 2: Number of Examples vs. Model Performance

		Tuning			Training					Validation				
NN	**Task**	**Weighed Samping**	**Weight Decay**	**Dropout**	**Loss**	**Accur**	**Prec**	**Recall**	**F1**	**Loss**	**Accur**	**Prec**	**Recall**	**F1**
ResNet18	**3-Class**	Yes	1.00E-02	0.2	3.04E-02	99.4	99.4	99.4	99.4	0.497	89.6	86.1	83.2	84.5
ResNet18	**Regression-Based**	No	0	0	2.07E-02	98.2	80.4	77.2	78.6	0.185	83.5	68.7	61.3	63.9
ResNet18	**6-Class**	No	0	0	4.24 e-2	98.3	95.9	95.9	95.9	0.939	79.9	68.4	61.3	63.9

		Tuning			Test				
NN	**Task**	**Weighed Samping**	**Weight Decay**	**Dropout**	**Loss**	**Accur**	**Prec**	**Recall**	**F1**
ResNet18	**3-Class**	Yes	1.00E-02	0.2	0.504	86.4	75.1	76.1	75.5
ResNet18	**Regression-Based**	No	0	0	0.236	82.6	45.0	48.8	46.3
ResNet18	**6-Class**	No	0	0	0.935	75.3	48.0	48.4	48.1

Figure 3: Best Models for 6-Class, 3-Class, and Regression-Based Classification Tasks

training sets have the same proportion of images for each class, our overfitting problem can be attributed from having insufficient training data. Thus, we doubt that tuning hyperparameters will fix the overfitting problem alone. Nonetheless, for the 3-class and regression-based tasks, we still tried L2 regularization (regularization constants 1e-2 and 1e-3 respectively) and dropout (rate of 0.2) for the last one or two fully connected layers for our transfer learning models. By making the model less complex, regularization has to potential to reduce overfitting to the training set. By randomly dropping the weights of different neurons, dropout essentially averages between different neural networks, thus producing regularization effect.

From running 5 experiments, weighed sampling, weight decay (1e-3), and dropout (0.2) all degrade the performance of the ResNet18 and MobileNetV2 models. Weighed sampling essentially trains less using the Healthy class and training more on the minority classes. But since there are very few images in the minority class, this makes the model overfit more to those minority examples in the training set, but with so few examples, it is difficult to generalize well. Regularization showed no significant effect, and dropout might have resulted in a undertrained model since the performance on the training set decreased significantly while not generalizing well to the validation set.

We ran 14 experiments on ResNet18 and MobileNetV2 for the 3-class regression task. Weighed sampling degraded validation metrics for MobileNetV2, and it traded for higher recall and F1 scores with lower precision and accuracy for ResNet18. Dropout of 0.2 improved all validation metrics for ResNet18 while decreasing them all for MobileNetV2. Regularization seems to show slightly positive results for both models: MobileNetV2 (1e-2) and ResNet18 (1e-3).

5.3 Best Models and Discussion

Finally, for the three separate tasks, the models in Figure 3 performed the best on the validation set. While the accuracies are comparable between the validation and test set, the macro averaged F1 scores are much worse for the test set. Upon closer examination, the individual F1 scores are extremely low for minority classes: classes 3 and 4 have F1 scores of 0% for the regression-based classification task and 16.7% and 0% for the 6-class classification task, and the other non-Healthy classes have approximately F1 scores of 60-75%. The confusion matrix and breakout of the classes in each task looks similiar to the charts in Figure 2. (More detailed discussion exists in the Appendix Section 8.6.) Overall, this reinforces our previous hypothesis that the key problem lies in not having sufficient data for the minority classes, and thus hyperparameter tuning alone without getting more data will likely not make the models more generalizable.

6 Conclusion/Future Work

In the one-through approach, the FCNN did not perform well: it lacks complexity (hidden layers and units), and we were restricted to use a small dimension (64x64) to keep the input size (and number of parameters) manageable. With a small training set, the custom-made CNN could not learn the diseased pattern well enough for the classification task. The ResNet18 transfer learning model did the best since 1) the pretraining provided prior weights/knowledge helpful for our classification task 2) the 18-layer depth allowed for decent amount of complexity and learning of features. In the two-task approach, the 3-class classification task is a simplification of the original 6-class task, and since this reduces the insufficient data problem for minority rust level classes, we see increased accuracy (75.3% to 86.4%) and macro averaged F1 scores (48.1% to 75.5%) using best performing models on the test set. While regression-adopted ResNet18 model has higher accuracy on the regression-based task than the 6-class task, but it has slightly lower F1 score.

While we experienced problems of overfitting and class imbalance, the largest problem is the lack of training data. This results in decent accuracy but low F1 scores (especially for minority classes) in validation and test sets. While we tried training with weighed sampling, regularization, and dropout, we did not expect them to yield significant improvements.

Had we have the resources, we would collect more images from minority classes. Trying more hyperparameters or new NN architecture without more data will likely result in futile efforts. With more computation power, it would also be a good idea to use a larger resolution than 224x224. Additionally, since the diseased spots often take up very small portion of the leaf, performing segmentation first might be useful for the classification task. Finally, after collecting more data for the minority class, we would be interested in integrating the two tasks into one continuous one, such that each of the two tasks run on the set of the images and the outputs from them are combined to generate a single classification label.

7 Contributions

The CS230 PyTorch Vision example[7] provided the starter code for this project, although significant changes were made for 1) separate data processing procedures for each of the three classification tasks, 2) modification of the the fully connected neural network, 3) customized implementation of AlexNet, 4) transfer learning with AlexNet, ResNet18, and MobileNetV2, 5) evaluation metrics, 6) implementation of weighed sampling, and 7) hyperparameter tuning.

8 Appendix

8.1 Picture Examples of the Six Coffee Leaf Conditions

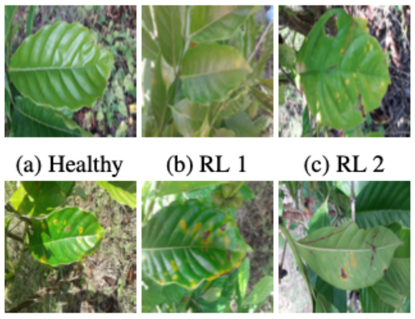


Figure 4: Sample image from RoCoLe

8.2 Activation Functions

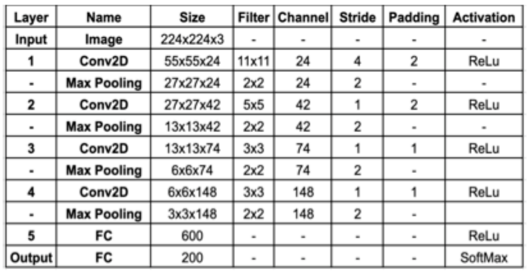
$$\text{ReLu}(z) = \max(0, z)$$

$$\text{ReLu6}(z) = \min(\max(0, z), 6)$$

$$\text{SoftMax: } g(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \text{ (and in our context} = \frac{e^{W_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{W_j^T x^{(i)}}})$$

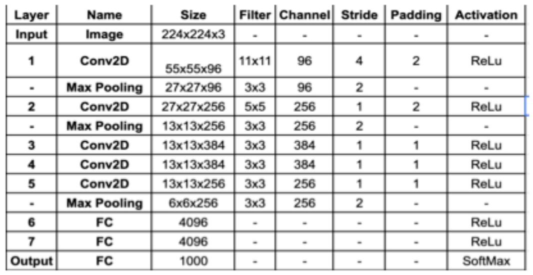
8.3 Neural Network Architecture Details

The follow section contains NN Architectures of models used in this project.



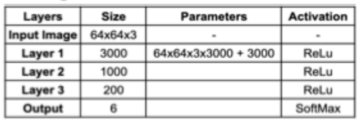
Layer	Name	Size	Filter	Channel	Stride	Padding	Activation
Input	Image	224x224x3	-	-	-	-	-
1	Conv2D	55x55x24	11x11	24	4	2	ReLu
-	Max Pooling	27x27x24	2x2	24	2	-	-
2	Conv2D	27x27x42	5x5	42	1	2	ReLu
-	Max Pooling	13x13x42	2x2	42	2	-	-
3	Conv2D	13x13x74	3x3	74	1	1	ReLu
-	Max Pooling	6x6x74	2x2	74	2	-	
4	Conv2D	6x6x148	3x3	148	1	1	ReLu
-	Max Pooling	3x3x148	2x2	148	2	-	
5	FC	600	-	-	-	-	ReLu
Output	FC	200	-	-	-	-	SoftMax

(a) AlexNet-Inspired CNN Architecture



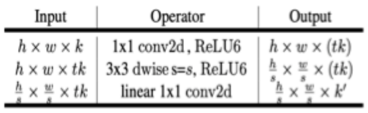
Layer	Name	Size	Filter	Channel	Stride	Padding	Activation
Input	Image	224x224x3	-	-	-	-	-
1	Conv2D	55x55x96	11x11	96	4	2	ReLu
-	Max Pooling	27x27x96	3x3	96	2	-	-
2	Conv2D	27x27x256	5x5	256	1	2	ReLu
-	Max Pooling	13x13x256	3x3	256	2	-	-
3	Conv2D	13x13x384	3x3	384	1	1	ReLu
4	Conv2D	13x13x384	3x3	384	1	1	ReLu
5	Conv2D	13x13x256	3x3	256	1	1	ReLu
-	Max Pooling	6x6x256	3x3	256	2	-	-
6	FC	4096	-	-	-	-	ReLu
7	FC	4096	-	-	-	-	ReLu
Output	FC	1000	-	-	-	-	SoftMax

(b) AlexNet Architecture Adopted for 224x224 Inputs



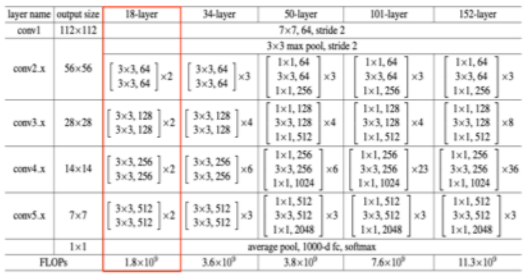
Layers	Size	Parameters	Activation
Input Image	64x64x3	-	-
Layer 1	3000	64x64x3x3000 + 3000	ReLu
Layer 2	1000		ReLu
Layer 3	200		ReLu
Output	6		SoftMax

(c) Fully Connected Neural Network Architecture



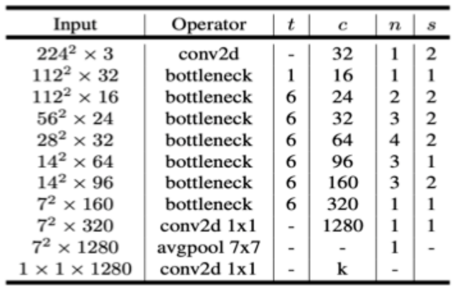
Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

(d) Bottleneck Residual Block for MobileNetV2[12]



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112		7×7, 64, stride 2			
			3×3 max pool, stride 2			
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1		average pool, 1000-d fc, softmax			
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

(a) ResNet18 Architecture[3]



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

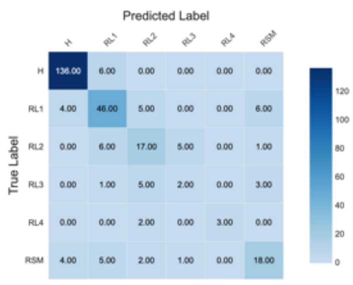
(b) MobileNetV2 Architecture[12]

Figure 6

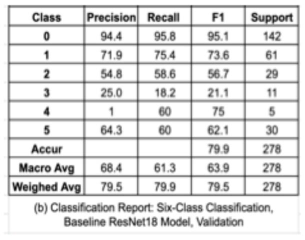
8.4 Results: Confusion Matrix and Classification Report

Since we have ran 5 baseline models for the 6-class classification task, 5 baseline models and 14 experiments for the 3-class classification task, and 3 baseline model and 5 experiments for the regression-based classification task, it will be unrealistic to include all the confusion matrices and classification reports here. However, since many of them share similar patterns, we will include a few additional examples here.

According to Figure 7.1.a, most of the incorrect labels are either along the green diagonal, in the last row, or in the last column. Incorrect labels near the green diagonal makes sense since it might be difficult to separate out the different rust level given their similarities. Mislabeling in the last row and column indicates that we are misidentifying another class as RSM, and we are misidentifying RSM as another class. This pattern seems to follow that RSM is a minority class with only 30 examples in the validation set (meaning the training set also has a low proportion of RSM examples). Additionally, we again see inaccurate labeling seems to correlate inversely porportional to the number of (training) examples, in Figure 7.1.b and 7.2.b. All of this suggests that we are insufficient amount of training data, and leads to low F1 scores despite accuracy being approximately 80%.



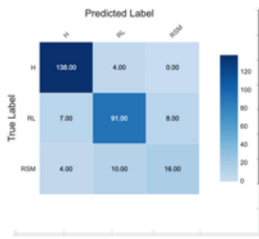
	H	RL1	RL2	RL3	RL4	RSM
H	136.00	6.00	0.00	0.00	0.00	0.00
RL1	4.00	46.00	5.00	0.00	0.00	6.00
RL2	0.00	6.00	17.00	5.00	0.00	1.00
RL3	0.00	1.00	5.00	2.00	0.00	3.00
RL4	0.00	0.00	2.00	0.00	3.00	0.00
RSM	4.00	5.00	2.00	1.00	0.00	18.00



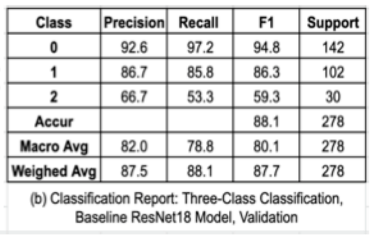
Class	Precision	Recall	F1	Support
0	94.4	95.8	95.1	142
1	71.9	75.4	73.6	61
2	54.8	58.6	56.7	29
3	25.0	18.2	21.1	11
4	1	60	75	5
5	64.3	60	62.1	30
Accur			79.9	278
Macro Avg	68.4	61.3	63.9	278
Weighed Avg	79.5	79.9	79.5	278

(b) Classification Report: Six-Class Classification, Baseline ResNet18 Model, Validation

(a) 1. Six-Class Classification Task ResNet18 Baseline



	H	RL	RSM
H	138.00	4.00	0.00
RL	7.00	91.00	8.00
RSM	4.00	10.00	16.00



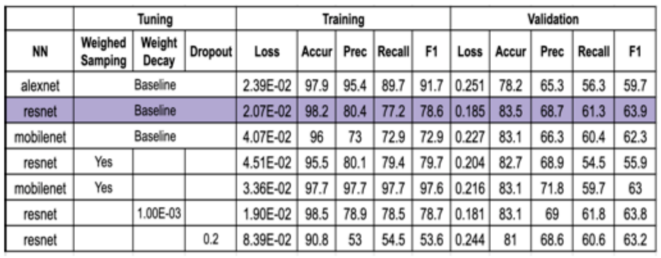
Class	Precision	Recall	F1	Support
0	92.6	97.2	94.8	142
1	86.7	85.8	86.3	102
2	66.7	53.3	59.3	30
Accur			88.1	278
Macro Avg	82.0	78.8	80.1	278
Weighed Avg	87.5	88.1	87.7	278

(b) Classification Report: Three-Class Classification, Baseline ResNet18 Model, Validation

(b) 2. Three-Class Classification ResNet18 Baseline

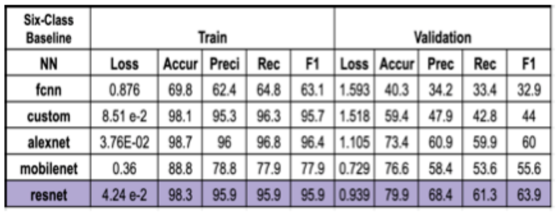
Figure 7

8.5 Baseline and Experiment Metrics: Loss, Accuracy, and Macro Averages Precision, Recall, and F1



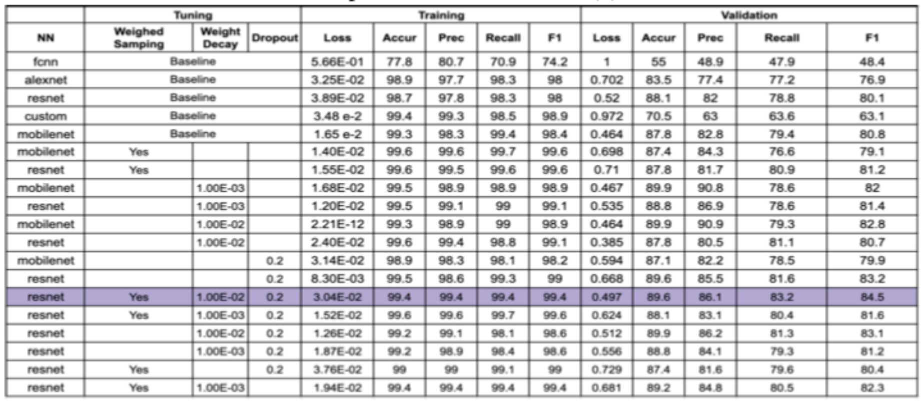
	Tuning			Training					Validation				
NN	Weighed Samping	Weight Decay	Dropout	Loss	Accur	Prec	Recall	F1	Loss	Accur	Prec	Recall	F1
alexnet	Baseline			2.39E-02	97.9	95.4	89.7	91.7	0.251	78.2	65.3	56.3	59.7
resnet	Baseline			2.07E-02	98.2	80.4	77.2	78.6	0.185	83.5	68.7	61.3	63.9
mobilenet	Baseline			4.07E-02	96	73	72.9	72.9	0.227	83.1	66.3	60.4	62.3
resnet	Yes			4.51E-02	95.5	80.1	79.4	79.7	0.204	82.7	68.9	54.5	55.9
mobilenet	Yes			3.36E-02	97.7	97.7	97.7	97.6	0.216	83.1	71.8	59.7	63
resnet		1.00E-03		1.90E-02	98.5	78.9	78.5	78.7	0.181	83.1	69	61.8	63.8
resnet			0.2	8.39E-02	90.8	53	54.5	53.6	0.244	81	68.6	60.6	63.2

(a) Regression-Based Classification Task Experiments



Six-Class Baseline	Train					Validation				
NN	Loss	Accur	Preci	Rec	F1	Loss	Accur	Prec	Rec	F1
fcnn	0.876	69.8	62.4	64.8	63.1	1.593	40.3	34.2	33.4	32.9
custom	8.51 e-2	98.1	95.3	96.3	95.7	1.518	59.4	47.9	42.8	44
alexnet	3.76E-02	98.7	96	96.8	96.4	1.105	73.4	60.9	59.9	60
mobilenet	0.36	88.8	78.8	77.9	77.9	0.729	76.6	58.4	53.6	55.6
resnet	4.24 e-2	98.3	95.9	95.9	95.9	0.939	79.9	68.4	61.3	63.9

(b) Six-Class Classification Task Baseline



	Tuning			Training					Validation				
NN	Weighed Samping	Weight Decay	Dropout	Loss	Accur	Prec	Recall	F1	Loss	Accur	Prec	Recall	F1
fcnn	Baseline			5.66E-01	77.8	80.7	70.9	74.2	1	55	48.9	47.9	48.4
alexnet	Baseline			3.25E-02	98.9	97.7	98.3	98	0.702	83.5	77.4	77.2	76.9
resnet	Baseline			3.89E-02	98.7	97.8	98.3	98	0.52	88.1	82	78.8	80.1
custom	Baseline			3.48 e-2	99.4	99.3	98.5	98.9	0.972	70.5	63	63.6	63.1
mobilenet	Baseline			1.65 e-2	99.3	98.3	99.4	98.4	0.464	87.8	82.8	79.4	80.8
mobilenet	Yes			1.40E-02	99.6	99.6	99.7	99.6	0.698	87.4	84.3	76.6	79.1
resnet	Yes			1.55E-02	99.6	99.5	99.6	99.6	0.71	87.8	81.7	80.9	81.2
mobilenet		1.00E-03		1.68E-02	99.5	98.9	98.9	98.9	0.467	89.9	90.8	78.6	82
resnet		1.00E-03		1.20E-02	99.5	99.1	99	99.1	0.535	88.8	86.9	78.6	81.4
mobilenet		1.00E-02		2.21E-12	99.3	98.9	99	98.9	0.464	89.9	90.9	79.3	82.8
resnet		1.00E-02		2.40E-02	99.6	99.4	98.8	99.1	0.385	87.8	80.5	81.1	80.7
mobilenet			0.2	3.14E-02	98.9	98.3	98.1	98.2	0.594	87.1	82.2	78.5	79.9
resnet			0.2	8.30E-03	99.5	98.6	99.3	99	0.668	89.6	85.5	81.6	83.2
resnet	Yes	1.00E-02	0.2	3.04E-02	99.4	99.4	99.4	99.4	0.497	89.6	86.1	83.2	84.5
resnet	Yes	1.00E-03	0.2	1.52E-02	99.6	99.6	99.7	99.6	0.624	88.1	83.1	80.4	81.6
resnet		1.00E-02	0.2	1.26E-02	99.2	99.1	98.1	98.6	0.512	89.9	86.2	81.3	83.1
resnet		1.00E-03	0.2	1.87E-02	99.2	98.9	98.4	98.6	0.556	88.8	84.1	79.3	81.2
resnet	Yes		0.2	3.76E-02	99	99	99.1	99	0.729	87.4	81.6	79.6	80.4
resnet	Yes	1.00E-03		1.94E-02	99.4	99.4	99.4	99.4	0.681	89.2	84.8	80.5	82.3

(c) Three-Class Classification Task Experiments

Figure 8

8.6 Deciding the Initial Hyperparameters

The implementation was modeled after the CS230 PyTorch Vision example[7]. Since the initial training error was high, we used a higher epoch number of 30 (instead of 10) to allow for longer training, and learning rate was decreased to 1e-4. Since we have high bias, we did not use dropout because we wanted to fit the training set well first. The batch size of 32 was kept the same as the PyTorch example.

9 Code Access

The full code repository can be viewed here: https://github.com/shellydeng/cs230-project-coffee-leaf-disease.

10 References

1) Buda, Mateusz, et al. “A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks.” Neural Networks, Pergamon, 29 July 2018, www.sciencedirect.com/science/article/abs/pii/S0893608018302107.

2) Fuentes, Alvaro, et al. “A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition.” MDPI, Multidisciplinary Digital Publishing Institute, 4 Sept. 2017, www.mdpi.com/1424-8220/17/9/2022.

3) He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” Deep Residual Learning for Image Recognition - IEEE Conference Publication, 27 Dec. 2016, ieeexplore.ieee.org/document/7780459.

4) Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” ResearchGate, Jan. 2012, www.researchgate.net/publication/267960550_ImageNet_Classification_with_Deep_Convolutional_Neural_Networks.

5) Krizhevsky, Alex, et al. “One Weird Trick for Parallelizing Convolutional Neural Networks.” ArXiv.org, 26 Apr. 2014, arxiv.org/abs/1404.5997.

6) LeCun, Yann, et al. “Gradient-Based Learning Applied to Document Recognition.” Gradient-Based Learning Applied to Document Recognition - IEEE Journals amp; Magazine, Nov. 1998, ieeexplore.ieee.org/document/726791.

7) Nair, Surag, et al. CS230-Stanford CS230 Code Examples, (2019), GitHub repository, https://github.com/cs230-stanford/cs230-code-examples/tree/master/pytorch/vision

8) Orozco, Ismael, and María E. Buemi. “A Study on Pedestrian Detection Using a Deep Convolutional Neural Network.” A Study on Pedestrian Detection Using a Deep Convolutional Neural Network - IET Conference Publication, 20 Apr. 2016, ieeexplore.ieee.org/document/7777677.

9) Parraga-Alava, Jorge, et al. “RoCoLe: A Robusta Coffee Leaf Images Dataset for Evaluation of Machine Learning Based Methods in Plant Diseases Recognition.” Data in Brief, Elsevier, 19 Aug. 2019, www.sciencedirect.com/science/article/pii/S2352340919307693.

10) Paszke, Adam, et al. “Automatic Differentiation in PyTorch.” OpenReview, 28 Oct. 2017, openreview.net/forum?id=BJJsrmfCZ.

11) Pedregosa, Fabian, et al. “Scikit-Learn: Machine Learning in Python.” Journal of Machine Learning Research, 1 Jan. 1970, jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

12) Sandler, Mark, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” MobileNetV2: Inverted Residuals and Linear Bottlenecks - IEEE Conference Publication, 18 June 2018, ieeexplore.ieee.org/document/8578572.

13) Sorte, Lucas Ximenes Boa, et al. “Coffee Leaf Disease Recognition Based on Deep Learning and Texture Attributes.” Procedia Computer Science, Elsevier, 14 Oct. 2019, www.sciencedirect.com/science/article/pii/S1877050919313468.

14) Wallelign, Serawork. “An Intelligent System for Coffee Grading and Disease Identification.” Research Gate, Feb. 2020, www.researchgate.net/publication/339946616_An_Intelligent _System_for_Coffee_Grading_and_Disease_Identification.