What Kind of Sick Tomato are You?

Identifying Tomato Diseases Using Image Classification Models

Yixi Yang, John Scott, Patricia Gallagher

08/02/2022

1) Problem Motivation

Plant diseases are problematic for growers on both individual and commercial levels. Plant pathogens alone are estimated to be responsible for 16% of global yield losses. Identifying plant diseases is essential for productive growing because without proper identification, plants cannot be efficiently or effectively treated, leading to larger yield losses and wasted resources. However, identifying plant illnesses can be incredibly difficult due to their vast variety and applying cures blindly can be expensive, ineffective, and dangerous. This project aims to develop an algorithm for identifying plant diseases from an input image of a plant leaf. Within the scope of this report, our dataset will only include tomato leaves and diseases, for a total of ten classes. Focusing this report on tomato plants was prudent because of the large amount of data available on tomato leaves/diseases and because of tomato's extensive impact in the agricultural sector. (In 2019, tomatoes were the second most consumed vegetable in the United States according to the USDA.) While this study focuses on tomatoes, the ultimate goal is to create a flexible algorithm that will be applicable to other plant types if trained accordingly. In the future, this algorithm can be used in mobile applications by amatuer plant-growers.

2) Dataset Description

The original dataset was published in 2019 by Mendeley Data and included 61,486 256x256 images. The images were categorized into thirty-nine different classes of leaf types and diseases. Each image displays a plant leaf against differing backgrounds. The original publishers of the data applied different augmentation techniques to the dataset with the purpose of increasing the dataset size. The different techniques include image flipping, Gamma correction, noise injection, PCA color augmentation, rotation, and scaling. There were at least 1,000 images for each class post augmentation.

Since this report focuses on tomato leaves and diseases, the original dataset was filtered from thirty-nine classes to include only the ten tomato classes, with nine different disease classes and one healthy class. The original dataset includes an unbalanced amount of data points per tomato disease category. The bacterial spot class had over 2,000 data points while the mosaic virus class and early blight class only had

1,000. To balance the dataset, each tomato disease category was limited to one thousand data points, resulting in a final dataset with ten thousand data points. The data points for each class were selected randomly.

Bacterial Spot	Early Blight	Healthy	Late Blight	Leaf Mold	Septoria	Spider Mites	Target Spot	Mosaic Virus	Yellow Leaf Curl
2127	1000	1591	1908	1000	1771	1676	1404	1000	5357

3) Solution/Approach Description

To clean and prepare the data for analysis, we decided to keep the images at 256x256 size for our first model iterations and compress them to smaller sizes in later iterations of model-building to see whether image sizes made a difference in model accuracy. We kept the images at the highest available size initially because most of our images have large areas of green color, with small differences in yellow or dark patches in the leaves, so we were afraid that compressing images would remove the small dark spots indicating the disease.

We then decided to use 20% of the data for testing, 10% for validation within our training segment, and the remaining 70% for our training data. Since we have already balanced our dataset such that there were 1,000 randomly selected images for each class, taking different percentages of the data to train, validate, or test should yield similar distributions of classes in all three segments.

Finally, we applied additional data augmentation transformations to our training set. We increased the brightness and contrast of our images slightly by applying a delta of 0.1 and a contrast factor of 2 to our training dataset. We also applied a random flipping method so that some images are flipped on the left and right. We then added the augmented training images to our original training dataset such that we had a total of 14,000 training images.

The original publisher of our data had created a 9-layer deep convolutional neural network to classify images into all 39 possible categories (Pandian & Geetharamani, 2019). We wanted to explore a similar approach in addition to simpler models, such as logistic regression models and feedforward neural network models.

Our approach included four different types of models:

1. Logistic regression
2. Feedforward neural network
3. Convolutional neural network using our own chosen parameters
4. Convolutional neural network based on AlexNet's architecture, trained from scratch

5. Convolutional neural network transfer learning using VGG16 and VGG19, retraining performed with two dense layers

We used different image sizes and varied parameters for each of the model types and observed the training accuracy, validation accuracy, and training time to decide the best models. We then employed ensemble testing and picked five of the best models and calculated equal-weight averages of their results to calculate our final results. This ensemble design was what we used to run our testing data.

To simplify the creation, fitting, and timing of these models, a very simplified syntax for neural net training was made. The neural nets themselves were described as lists of lists, with the relevant numerical parameters in sequence. The keywords in this simple syntax were “flatten,” “dense,” “conv,” ‘max_pool,” “batch_norm,” “dropout,” “VGG16,” “VGG19,” and “output.” As an example, a convolutional layer takes this form: [‘conv’, KERNEL_SIZE, STRIDE, NUM_FILTERS]. Along with a simple training syntax, this format allowed the creation and testing of many different parameters and neural net structures using dictionary comprehensions.

4) Experiments

For all the models, we used the Adam optimizer with a learning rate of 0.001. We calculated the loss using the sparse categorical cross-entropy since our results had multiple classes. The final output layer was always a softmax layer. We also counted the training time for each model, as well as the training and validation accuracies. We used a batch size of 125 for all our model evaluations. Length of training time was recorded as well as validation accuracy. The number of parameters was not directly recorded for each system.

For the logistic regression modeling, we flattened the images and then applied a softmax activation function to calculate probabilities that the image belonged to each of the ten possible classes. The class with the highest probability is our predicted class for the image. We created five different iterations of our logistic model with different image sizes ranging between 256x256 and 16x16.

ID	type	subID	epochs	nn_struct	img_size
0	logistic	0	10	[['flatten'], ['output', 10]]	256
1	logistic	1	10	[['flatten'], ['output', 10]]	128
2	logistic	2	10	[['flatten'], ['output', 10]]	64
3	logistic	3	10	[['flatten'], ['output', 10]]	32
4	logistic	4	10	[['flatten'], ['output', 10]]	16

For the feedforward neural networks, we tested models with between one and four hidden dense layers in the model. The number of neurons in each dense layer was varied across many iterations. We also added an output layer for the ten categories. We fed each model iteration images of sizes 256x256, 128x128, and 64x64.

ID	type	subID	epochs	nn_struct	img_size
33	FF_4	3	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128], ['output', 10]]	64
35	FF_4	5	10	[['flatten'], ['dense', 256], ['dense', 128], ['dense', 64], ['dense', 32], ['output', 10]]	64
13	FF_1	8	10	[['flatten'], ['dense', 512], ['output', 10]]	64
12	FF_1	7	10	[['flatten'], ['dense', 1024], ['output', 10]]	64
30	FF_4	0	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128], ['output', 10]]	128

For our first iterations of CNN models, we tried four main structures: (1) one convolutional layer and pooling layer followed by one hidden layer, (2) two convolutional layers and pooling layers followed by one hidden layer, (3) one convolutional and pooling layers followed by two hidden layers, (4) and two convolutional and pooling layers followed by two hidden layers. We experimented with different kernel sizes, strides, and number filters within each main structure, as well as number of nodes in the hidden layers and size of images fed through the model.

ID	type	subID	epochs	nn_struct	img_size
69	CNN_2_1	1	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128
95	CNN_2_2	15	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64
92	CNN_2_2	12	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64
85	CNN_2_2	5	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64
70	CNN_2_1	2	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128

For the models built with inspiration from existing models, we hand-built one based off of AlexNet, with simplifications in the final dense layers where we reduced the number of neurons from 4,096 to 1,024. For VGG16 and VGG19, we used transfer learning, where we removed the final layers and retrained on our data. The VGG16-based model was trained with 128x128-sized images using 5 epochs. The two final hidden layers had 512 neurons, then 64 neurons. The VGG19-based model was trained with 64x64 images using 5 epochs, with two final hidden layers of 128 neurons and then 32 neurons.

ID	type	subID	epochs	nn_struct	img_size
98	ALEXNIET	0	10	[['conv', 11, 4, 96], ['max_pool', 3], ['conv', 5, 1, 256], ['max_pool', 3], ['conv', 3, 1, 384], ['conv', 3, 1, 384], ['max_pool', 3], ['flatten'], ['dropout', 0.5], ['dense', 1024], ['dropout', 0.5], ['dense', 1024], ['output', num_outputs]]	256
99	VGG_16	0	5	[['VGG16'], ['dense', 512], ['dense', 64], ['output', 10]]	128
100	VGG_19	0	5	[['VGG19'], ['dense', 128], ['dense', 32], ['output', 10]]	64
101	VGG_19	0	8	[['VGG19'], ['dense', 1024], ['dense', 256], ['output', 10]]	256

We also wanted to see if prediction results would improve with ensemble testing. For the sake of training time, we took the five fastest CNNs that had tolerably good accuracy results, and equal-weighted them to produce a final set of predictions.

By varying 1) neural structure, 2) image size, and 3) the number of epochs, 102 unique neural net training conditions were performed. A full list of all conditions appears in the Appendix of this document, but as a summary, this table shows the relative number of each type:

NN Type	Number of trained NNs	Total Training Time (min)
Logistic	5	2.42
FF, 1 layer	9	94.28
FF, 2 layer	6	30.39
FF, 3 layer	10	10.15
FF, 4 layer	6	6.96
CNN (1 conv, 1 dense)	32	191.60
CNN (2 conv, 1 dense)	8	168.81
CNN (1 conv, 2 dense)	4	5.66
CNN (2 conv, 2 dense)	18	203.82
Alex Net* (untrained)	1	52.31
VGG16 Transfer Learning	1	49.00
VGG19 Transfer Learning	2	528.00

*AlexNet structure was emulated and trained; while the overall structure was adapted, the weights were trained from scratch. An exact account of the structure can be found in the Appendix

5) Results

Logistic regression

The results of our five logistic regression models are shown below. These models all took very little time to train. Interestingly, the largest image sizes did not yield the highest validation accuracy. Our validation accuracy peaked at around 57% for a model with images sizes of 32x32.

ID	type	subID	epochs	nn_struct	img_size	final_train_a	final_valid_a	best_valid_a	training_time
0	logistic	0	10	[['flatten'], ['output', 10]]	256	0.6148	0.4360	0.4850	1.8869
1	logistic	1	10	[['flatten'], ['output', 10]]	128	0.6416	0.3670	0.5190	0.3316
2	logistic	2	10	[['flatten'], ['output', 10]]	64	0.7088	0.5040	0.5470	0.1117
3	logistic	3	10	[['flatten'], ['output', 10]]	32	0.6366	0.5720	0.5750	0.0528
4	logistic	4	10	[['flatten'], ['output', 10]]	16	0.5637	0.5000	0.5130	0.0381

Feedforward neural network

The results of our feedforward neural networks are shown as below. The best accuracy came from models with four hidden layers, although these were not much higher than those with just one hidden layer and hover around the low 70%. The most shocking discovery for us is that FFNNs with two and three dense layers had exceedingly low validation accuracies. Our FFNNs with two hidden layers only had validation accuracies in the low-teens percentage range, or even lower. Our FFNNs with three hidden layers only showed validation accuracies in the 30%-50% range, which is even worse than a simple logistic regression. Additionally, the number of nodes in the last hidden layer seem to have some correlation with validation accuracy: if the number

of nodes in the last hidden layer were too low, such as 64 or 32, then final accuracies were usually lower, holding other parameters constant.

ID	type	subID	epochs	nn_struct	img_size	final_train_ac	final_valid_a	best_valid_a	training_time
33	FF_4	3	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128], ['output', 10]]	64	0.8016	0.7280	0.7280	0.8938
35	FF_4	5	10	[['flatten'], ['dense', 256], ['dense', 128], ['dense', 64], ['dense', 32], ['output', 10]]	64	0.7409	0.7090	0.7090	0.2966
13	FF_1	8	10	[['flatten'], ['dense', 512], ['output', 10]]	64	0.7315	0.6750	0.6750	0.4858
12	FF_1	7	10	[['flatten'], ['dense', 1024], ['output', 10]]	64	0.7574	0.6580	0.6680	0.8306
30	FF_4	0	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128], ['output', 10]]	128	0.7871	0.6580	0.6580	2.7921

CNN (self-made structures)

We observed that CNN models immediately outperform previous models in terms of validation accuracies. The best models produced validation accuracies in the high-80%. Interestingly, adding more convolution layers seemed to improve validation accuracy but only by a few percentage points. Even the simplest CNN models produced validation accuracy over the mid-70%, suggesting that CNNs are inherently better for image classification than logistic models or FFNNs. Training times could vary greatly depending on size of kernels, number of filters, and number of hidden layer neurons. Typically, we found that increasing kernel size to an appropriate range and using a larger stride made running the models much more efficient.

ID	type	subID	epochs	nn_struct	img_size	final_train_ac	final_valid_a	best_valid_a	training_time
69	CNN_2_1	1	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.9770	0.8960	0.9070	9.9961
95	CNN_2_2	15	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.9817	0.8870	0.8870	27.1530
92	CNN_2_2	12	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.9749	0.8810	0.8810	9.2774
85	CNN_2_2	5	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.9809	0.8720	0.8790	5.6117
70	CNN_2_1	2	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.9799	0.8690	0.8690	9.8140

AlexNiet CNN - AlexNet copycat, weights trained from scratch

ID	type	subID	epochs	nn_struct	img_size	final_train_ac	final_valid_a	best_valid_a	training_time
98	ALEXNIET	0	10	[['conv', 11, 4, 96], ['max_pool', 3], ['conv', 5, 1, 256], ['max_pool', 3], ['conv', 3, 1, 384], ['conv', 3, 1, 384], ['conv', 3, 1, 384], ['max_pool', 3], ['flatten'], ['dropout', 0.5], ['dense', 1024], ['dropout', 0.5], ['dense', 1024], ['output', num_outputs]]	256	0.9330	0.8800	0.8800	52.3100

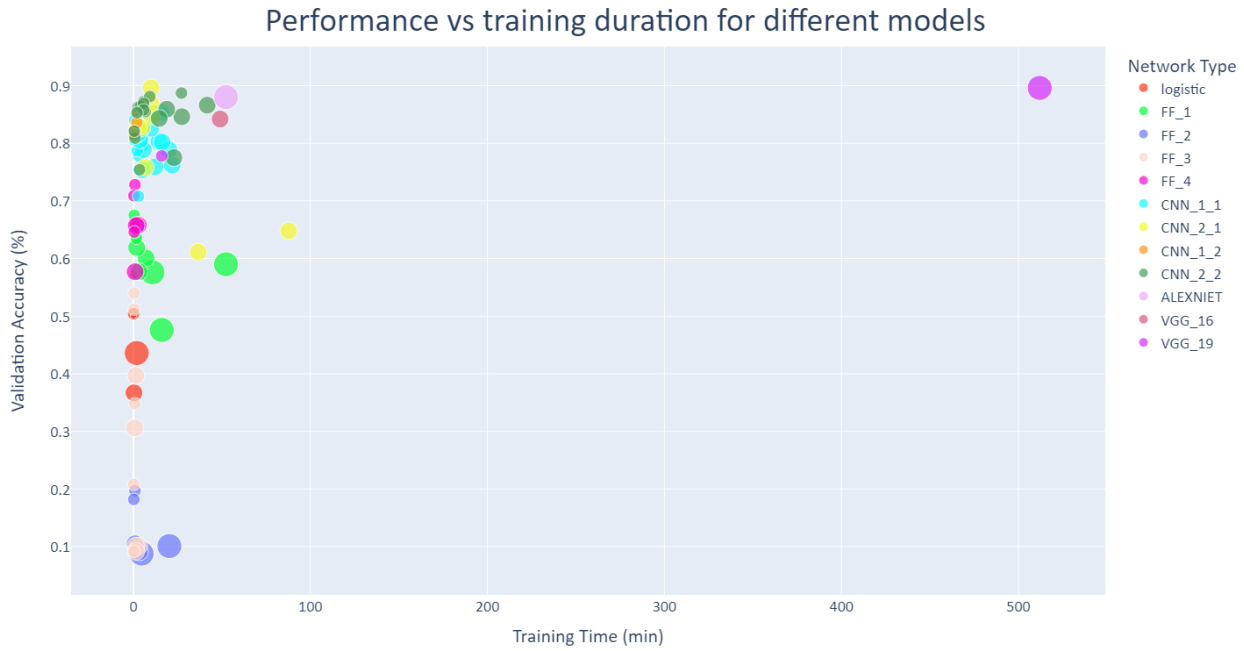
This AlexNet-like model differs in the number of neurons in the two final dense layers (1024 instead of 4096). The reasoning behind this change was two-fold: 1) the training time and number of parameters might have been unworkable on the machines available, and 2) the full AlexNet was used for classifying 1000 classes on ImageNet, rather than the 10 classes our limited CNN study was focused on. It was reasoned that far fewer final layer neurons would be required. This idea was not confirmed experimentally.

CNN (transfer learning) results

Our final models were built based on existing models with pretrained weights. The best performer of this group was VGG19 and two hidden layers with 1,024 neurons and 256 neurons, respectively. This model had final validation accuracy of nearly 90% and training accuracy of 99.9%. However, it also took the longest to run, by far, without yielding much difference to the “AlexNiet” model or some of our handmade CNN models.

ID	type	subID	epochs	nn_struct	img_size	final_train_ac	final_valid_a	best_valid_a	training_time
99	VGG_16	0	5	[['VGG16'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.9658	0.8420	0.8420	49.0000
100	VGG_19	0	5	[['VGG19'], ['dense', 128], ['dense', 32], ['output', 10]]	64	0.8030	0.7780	0.7780	16.0000
101	VGG_19	0	8	[['VGG19'], ['dense', 1024], ['dense', 256], ['output', 10]]	256	0.9990	0.8960	0.8960	512.0000

We made a graph of the training times and validation accuracies of all our models in an interactive plot, which would show the model parameters such as number of epochs, training time, accuracies, image size, and model structure. Although the screenshot below does not allow such hoverings, a “live” html version of the graph is included in the submitted materials, we can see that VGG_19, which tied as the most accurate model, took the longest time to run without much improvement. We can also see that the logistic models and FFNNs generally yielded lower validation accuracies than the CNNs.



Due to the single outlier (pretrained VGG19 with 256x256 images), a better view of the performance of the trained networks can be seen when the Training Time scale is

limited to 0→60 minutes.

The types of network that consistently showed high validation accuracy were the CNN groups, with very little or no gain observed training models beyond 10 minutes. While the single convolution, single dense CNNs did show overall good tradeoffs between time and accuracy, the CNNS with two convolutional layers and two dense layers consistently scored higher in accuracy.

We also wanted to see if prediction results would improve with ensemble testing. We ran the validation dataset on the five chosen CNNs and the ensemble model and compared the precisions of each model. The ensemble model did produce better precision, as shown by the precisions for each class and model below:

type	subID	epochs	nn_struct	img_size	final_train_ac	final_valid_a	best_valid_a	training_time
CNN_1_1	17	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.9876	0.8500	0.8500	12.3223
CNN_1_1	24	5	[['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.9796	0.8510	0.8510	4.4449
CNN_2_2	2	5	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.8959	0.8630	0.8630	2.5990
CNN_2_2	16	5	[['conv', 5, 2, 32], ['conv', 3, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.8359	0.8210	0.8210	0.4945
CNN_2_2	17	20	[['conv', 5, 2, 32], ['conv', 3, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.9886	0.8530	0.8820	2.0231

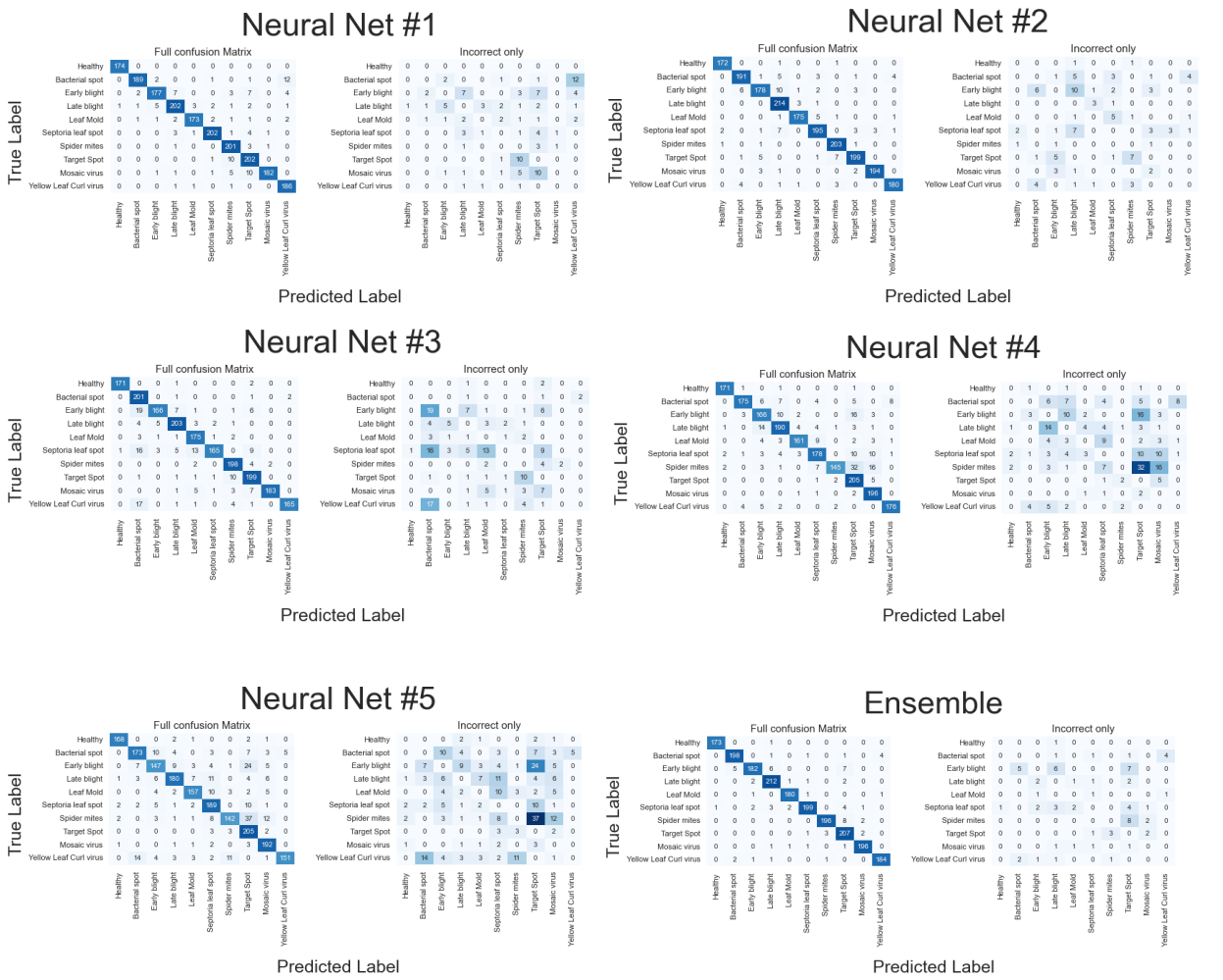
Precision numbers on test data are shown in the table below; each number from left to right is the precision of each label for the given model. The **bolded** numbers show the best performance as measured by precision.

CNN_1_1_17:	[0.99, 0.97, 0.93, 0.93, 0.99, 0.97, 0.86, 0.87, 1. , 0.96]
CNN_1_1_24:	[**1.0**, **0.99**, 0.94, 0.85, 0.96, 0.96, 0.92, **0.95**, **0.98**, 0.98]
CNN_2_2_2:	[**1.0**, 0.76, **0.95**, 0.85, 0.87, **0.99**, 0.9 , 0.86, **0.98**, **1.0**]
CNN_2_2_16:	[0.99, 0.93, 0.81, 0.84, 0.94, 0.86, 0.93, 0.7 , 0.87, 0.96]
CNN_2_2_17:	[0.98, 0.85, 0.85, 0.85, 0.86, 0.78, 0.91, 0.64, 0.83, **1.0**]
Ensemble:	[**1.0**, 0.98, **0.95**, **0.96**, **1.0** , **0.99**, **0.96**, 0.89, 0.96, **1.0**]

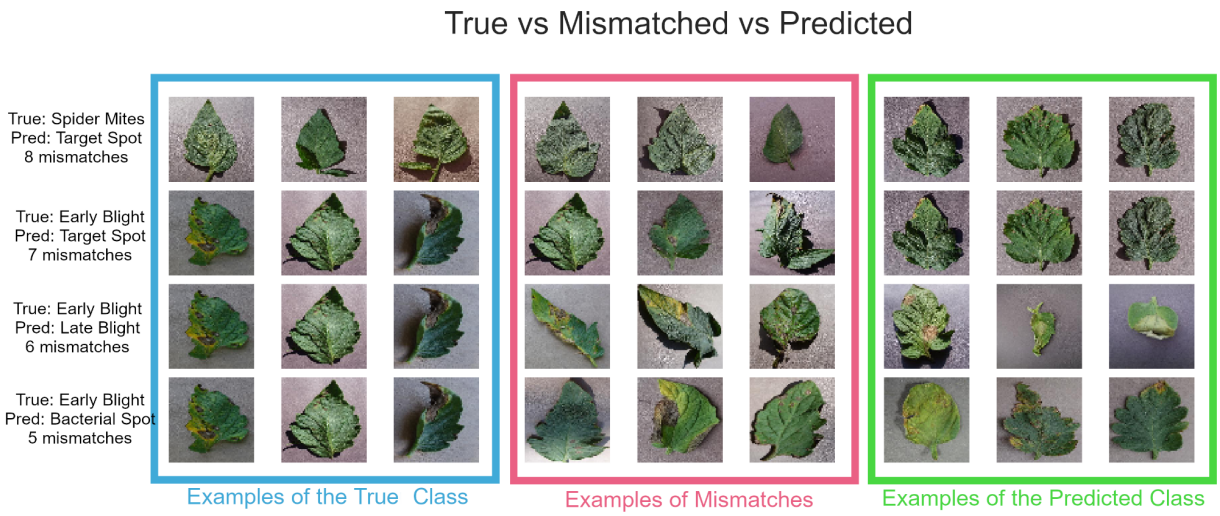
For all but three labels, the Ensemble model showed the highest precision; and in two of those three cases, the Ensemble model showed precision within 0.02 of the

best performing model. In only one case (label #7, Target Spot) did the Ensemble show significantly worse performance compared to the best of the individual models. For that particular label, the dispersion in precision was the widest.

We can further see from the confusion matrices of the five chosen CNNs and ensemble model below that the ensemble model had fewer misclassifications.



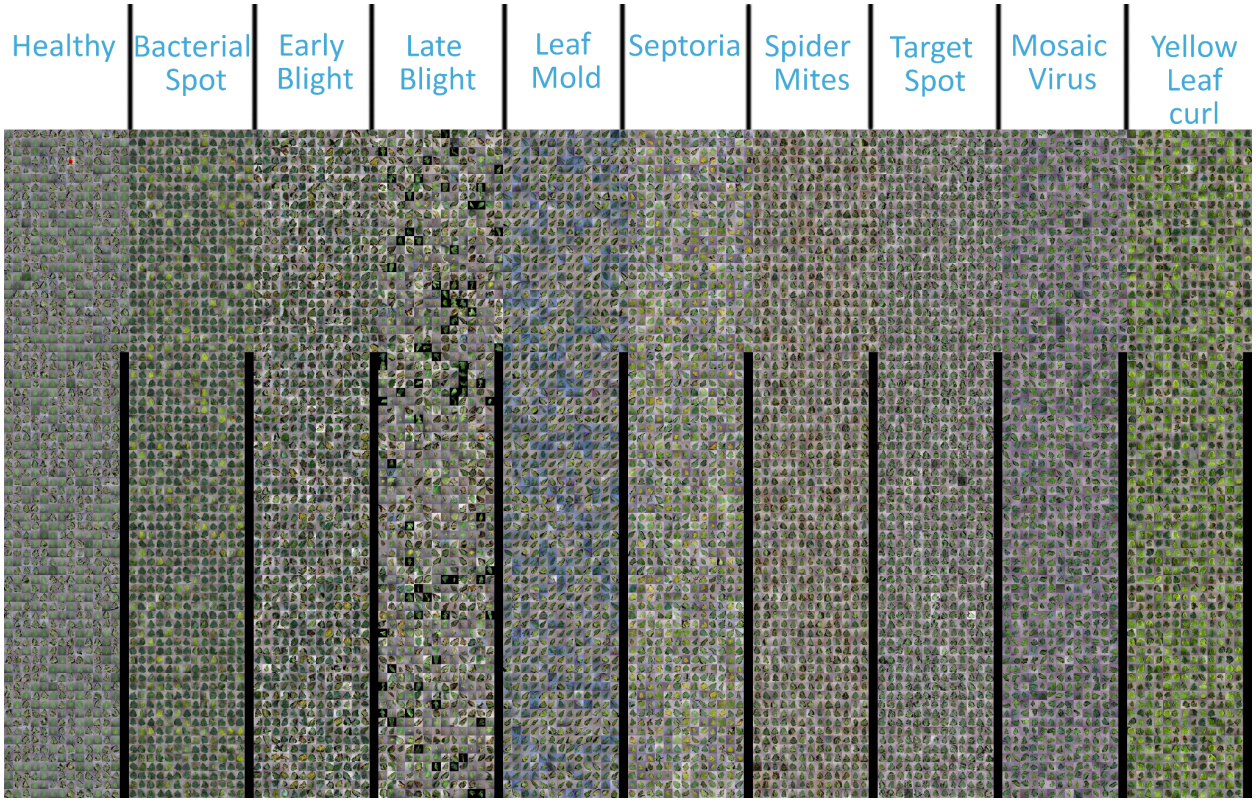
Given the above confusion matrices, the most frequent mistakes were evaluated visually, as seen in the graphic below. Each row is a type of misclassification as well as the number of mismatches. On the left side, boxed in blue, are examples of the True class; in the middle, boxed in red, are specific examples where a misclassification occurred; and on the right, boxed in green, are actual examples from the Predicted class. The purpose of this is visually check why certain classes may have been misclassified; and to see whether the misclassification makes sense from an intuitive level. In each case, there is a large degree of similarity visible by eye between the True and Predicted classes, with a common theme of browning and spotting for each.



6) Conclusions

In our experiments, convolutional neural networks yielded the best image classification results for our data. Even our simplest CNNs with one convolution layer and one dense layer outperformed the best FFNNs and logistic models. The best results were achieved from one of the simple CNNs as well as using transfer learning with VGG19, which were created using millions of images from ImageNet. We were able to get validation and testing accuracies near 90%. We also experimented with ensemble design, taking five high-speed and high-accuracy CNNs to average them for final weights, and the resulting predictions had higher precisions than the individual CNNs. Our models suggest that complexity of model structure does not necessarily correlate with better performance. Parameters such as number of nodes in FFNNs, kernel size and filters for CNNs, and the optimal batch size and image sizes made a greater difference in accuracy and training time.

One final note is about the quality and consistency of the source data. An ideal dataset should not show commonalities outside of the desired characteristics to be compared; in this case, it would be desirable to have consistent backgrounds in images, with the primary differences being the state of the leaves. However, in looking at all 10000 images simultaneously by shrinking them to fit in one image, general patterns heavily correlated to the label can be observed that do not relate to the diagnosis of a disease.



In this case, the backgrounds of the images are shown to have strong differences. The most notable difference in background appears in the Leaf Mold category, with a distinct blue hue very common in the background. This same blue hue is not visible, or at least not nearly as frequent, in any other class. As an additional example, the Late Blight group had the most black backgrounds, and the Spider Mites had the most brown backgrounds. It is likely that our classification algorithms are also using this biased background information, inflating our accuracy and precision numbers beyond the simple ability to discriminate between disease states of tomato plants. An idea on how to combat this plausible source of bias would be to first segment the leaf images, removing the background, and then perform the image classification task.

We believe that there are very little negative social impacts from our study, given the lack of direct involvement with human subjects. We are expecting a much larger positive impact since our model could help home gardeners across the country better identify diseases plaguing their produce and promote home gardening, eating fresh, and food waste reduction.

Future studies could include expanding our model to incorporate other types of vegetables to make it more universally useful for home gardeners. Furthermore, our study could be modified to focus on testing food freshness and edibility to further reduce food waste and improve local vegetable consumption.

Citations

Ficke A, Cowger C, Bergstrom G, Brodal G. Understanding Yield Loss and Pathogen Biology to Improve Disease Management: Septoria Nodorum Blotch - A Case Study in Wheat. Plant Dis. 2018 Apr;102(4):696-707. doi: 10.1094/PDIS-09-17-1375-FE. Epub 2018 Mar 5. PMID: 30673402.

J, ARUN PANDIAN; GOPAL, GEETHARAMANI (2019), “Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network”, Mendeley Data, V1, doi: 10.17632/tywbtsjrjv.1

Kantor, L., & Blazejczyk, A. (n.d.). Potatoes and tomatoes are the most commonly consumed vegetables. USDA ERS - Chart Detail. Retrieved from https://www.ers.usda.gov/data-products/chart-gallery/gallery/chart-detail/?chartId=58340

7) Appendix

Full training results are shown for all tested neural networks:

#	type	Num Epochs	Neural Structure Definition	Img Size	Final Train Acc	Final Val Acc	Best Val Acc	Training Time (min)
1	logistic	10	[['flatten'], ['output', 10]]	256	0.61	0.44	0.49	1.89
2	logistic	10	[['flatten'], ['output', 10]]	128	0.64	0.37	0.52	0.33
3	logistic	10	[['flatten'], ['output', 10]]	64	0.71	0.50	0.55	0.11
4	logistic	10	[['flatten'], ['output', 10]]	32	0.64	0.57	0.57	0.05
5	logistic	10	[['flatten'], ['output', 10]]	16	0.56	0.50	0.51	0.04
6	FF_1	10	[['flatten'], ['dense', 2048], ['output', 10]]	256	0.71	0.59	0.59	52.32
7	FF_1	10	[['flatten'], ['dense', 1024], ['output', 10]]	256	0.72	0.48	0.58	16.01
8	FF_1	10	[['flatten'], ['dense', 512], ['output', 10]]	256	0.72	0.58	0.58	10.78
9	FF_1	10	[['flatten'], ['dense', 2048], ['output', 10]]	128	0.73	0.60	0.60	7.12
10	FF_1	10	[['flatten'], ['dense', 1024], ['output', 10]]	128	0.75	0.58	0.58	3.16
11	FF_1	10	[['flatten'], ['dense', 512], ['output', 10]]	128	0.72	0.62	0.62	1.85
12	FF_1	10	[['flatten'], ['dense', 2048], ['output', 10]]	64	0.76	0.63	0.63	1.73
13	FF_1	10	[['flatten'], ['dense', 1024], ['output', 10]]	64	0.76	0.66	0.67	0.83
14	FF_1	10	[['flatten'], ['dense', 512], ['output', 10]]	64	0.73	0.68	0.68	0.49
15	FF_2	5	[['flatten'], ['dense', 2048], ['dense', 128], ['output', 10]]	256	0.11	0.10	0.10	20.32
16	FF_2	5	[['flatten'], ['dense', 512], ['dense', 32], ['output', 10]]	256	0.10	0.09	0.10	4.64
17	FF_2	5	[['flatten'], ['dense', 2048], ['dense', 128], ['output', 10]]	128	0.17	0.09	0.20	3.38
18	FF_2	5	[['flatten'], ['dense', 512], ['dense', 32], ['output', 10]]	128	0.11	0.10	0.11	0.93
19	FF_2	5	[['flatten'], ['dense', 2048], ['dense', 128], ['output', 10]]	64	0.19	0.20	0.23	0.88
20	FF_2	5	[['flatten'], ['dense', 512], ['dense', 32], ['output', 10]]	64	0.19	0.18	0.18	0.24
21	FF_3	5	[['flatten'], ['dense', 2048], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.10	0.10	0.10	2.94
22	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.11	0.10	0.11	1.42
23	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 256], ['dense', 64], ['output', 10]]	128	0.50	0.40	0.43	1.41
24	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 256], ['dense', 32], ['output', 10]]	128	0.10	0.09	0.09	1.40
25	FF_3	5	[['flatten'], ['dense', 512], ['dense', 128], ['dense', 32], ['output', 10]]	128	0.28	0.31	0.31	0.80
26	FF_3	5	[['flatten'], ['dense', 2048], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.42	0.35	0.37	0.81
27	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.57	0.54	0.54	0.40
28	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.57	0.51	0.52	0.38
29	FF_3	5	[['flatten'], ['dense', 1024], ['dense', 256], ['dense', 32], ['output', 10]]	64	0.10	0.09	0.10	0.38
30	FF_3	5	[['flatten'], ['dense', 512], ['dense', 128], ['dense', 32], ['output', 10]]	64	0.20	0.21	0.21	0.21
31	FF_4	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128],	128	0.79	0.66	0.66	2.79
32	FF_4	10	[['flatten'], ['dense', 512], ['dense', 256], ['dense', 128], ['dense', 64],	128	0.75	0.66	0.66	1.55
33	FF_4	10	[['flatten'], ['dense', 256], ['dense', 128], ['dense', 64], ['dense', 32],	128	0.69	0.58	0.58	0.96
34	FF_4	10	[['flatten'], ['dense', 1024], ['dense', 512], ['dense', 256], ['dense', 128],	64	0.80	0.73	0.73	0.89
35	FF_4	10	[['flatten'], ['dense', 512], ['dense', 256], ['dense', 128], ['dense', 64],	64	0.80	0.65	0.68	0.47
36	FF_4	10	[['flatten'], ['dense', 256], ['dense', 128], ['dense', 64], ['dense', 32],	64	0.74	0.71	0.71	0.30
37	CNN_1_1	5	[['conv', 9, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.91	0.76	0.76	21.89
38	CNN_1_1	5	[['conv', 9, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.92	0.80	0.80	14.36
39	CNN_1_1	5	[['conv', 9, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.92	0.80	0.80	5.17
40	CNN_1_1	5	[['conv', 9, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.93	0.78	0.79	3.05
41	CNN_1_1	5	[['conv', 9, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.93	0.79	0.80	20.03
42	CNN_1_1	5	[['conv', 9, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.90	0.76	0.78	12.16
43	CNN_1_1	5	[['conv', 9, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.89	0.81	0.81	4.76
44	CNN_1_1	5	[['conv', 9, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.90	0.71	0.74	2.83
45	CNN_1_1	5	[['conv', 9, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.95	0.83	0.83	5.05
46	CNN_1_1	5	[['conv', 9, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.92	0.75	0.78	5.26
47	CNN_1_1	5	[['conv', 9, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.93	0.82	0.82	1.28
48	CNN_1_1	5	[['conv', 9, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.91	0.82	0.82	1.08
49	CNN_1_1	5	[['conv', 9, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.94	0.79	0.80	5.69
50	CNN_1_1	5	[['conv', 9, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.94	0.82	0.83	2.82

51	CNN_1_1	5	[['conv', 9, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.90	0.81	0.81	1.22
52	CNN_1_1	5	[['conv', 9, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.88	0.82	0.82	0.76
53	CNN_1_1	5	[['conv', 5, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.98	0.85	0.85	15.86
54	CNN_1_1	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.99	0.85	0.85	12.32
55	CNN_1_1	5	[['conv', 5, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.96	0.84	0.86	4.35
56	CNN_1_1	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.97	0.85	0.85	2.82
57	CNN_1_1	5	[['conv', 5, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.97	0.80	0.83	16.21
58	CNN_1_1	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.97	0.83	0.85	9.79
59	CNN_1_1	5	[['conv', 5, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.96	0.85	0.85	4.23
60	CNN_1_1	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.94	0.79	0.79	2.11
61	CNN_1_1	5	[['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.98	0.85	0.85	4.44
62	CNN_1_1	5	[['conv', 5, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.96	0.83	0.83	2.56
63	CNN_1_1	5	[['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.95	0.83	0.83	1.13
64	CNN_1_1	5	[['conv', 5, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	64	0.90	0.81	0.81	0.72
65	CNN_1_1	5	[['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.98	0.81	0.81	3.73
66	CNN_1_1	5	[['conv', 5, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.94	0.82	0.82	2.21
67	CNN_1_1	5	[['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.90	0.83	0.83	0.99
68	CNN_1_1	5	[['conv', 5, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	64	0.91	0.84	0.84	0.71
69	CNN_2_1	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 1024], ['output', 10]]	128	0.98	0.85	0.88	10.26
70	CNN_2_1	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.98	0.90	0.91	10.00
71	CNN_2_1	10	[['conv', 9, 2, 64], ['max_pool', 2], ['conv', 5, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.98	0.87	0.87	9.81
72	CNN_2_1	5	[['conv', 7, 2, 32], ['conv', 3, 1, 32], ['flatten'], ['dense', 512], ['output', 10]]	128	0.97	0.76	0.76	6.74
73	CNN_2_1	5	[['conv', 7, 2, 32], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.96	0.83	0.83	4.83
74	CNN_2_1	5	[['conv', 7, 2, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['output', 10]]	128	0.92	0.83	0.83	2.78
75	CNN_2_1	5	[['conv', 7, 1, 64], ['conv', 5, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.87	0.65	0.65	87.76
76	CNN_2_1	5	[['conv', 7, 1, 32], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['output', 10]]	128	0.94	0.61	0.63	36.63
77	CNN_1_2	5	[['conv', 7, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 128], ['output', 10]]	64	0.94	0.85	0.85	2.41
78	CNN_1_2	5	[['conv', 7, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 128], ['output', 10]]	64	0.92	0.82	0.82	0.70
79	CNN_1_2	5	[['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 128], ['output', 10]]	64	0.97	0.84	0.84	1.97
80	CNN_1_2	5	[['conv', 5, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 128], ['output', 10]]	64	0.92	0.81	0.81	0.59
81	CNN_2_2	5	[['conv', 5, 2, 64], ['conv', 3, 2, 64], ['max_pool', 2], ['flatten'], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.86	0.81	0.81	0.93
82	CNN_2_2	5	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.89	0.75	0.79	3.44
83	CNN_2_2	5	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.90	0.86	0.86	2.60
84	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.97	0.86	0.86	6.66
85	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.99	0.86	0.89	4.83
86	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.98	0.87	0.88	5.61

87	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.98	0.86	0.88	3.90
88	CNN_2_2	10	[['conv', 7, 1, 64], ['max_pool', 2], ['conv', 3, 1, 64], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 128], ['output', 10]]	128	0.99	0.87	0.87	41.71
89	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.99	0.85	0.86	27.32
90	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.99	0.86	0.86	18.83
91	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.97	0.77	0.77	22.85
92	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.99	0.84	0.87	14.59
93	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.97	0.88	0.88	9.28
94	CNN_2_2	10	[['conv', 7, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.98	0.87	0.87	5.79
95	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 5, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.97	0.86	0.86	5.81
96	CNN_2_2	10	[['conv', 5, 1, 32], ['max_pool', 2], ['conv', 3, 1, 32], ['max_pool', 2], ['flatten'], ['dense', 512], ['dense', 64], ['output', 10]]	64	0.98	0.89	0.89	27.15
97	CNN_2_2	5	[['conv', 5, 2, 32], ['conv', 3, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.84	0.82	0.82	0.49
98	CNN_2_2	20	[['conv', 5, 2, 32], ['conv', 3, 2, 32], ['max_pool', 2], ['flatten'], ['dense', 256], ['dense', 64], ['output', 10]]	64	0.99	0.85	0.88	2.02
99	ALEXNIET	10	[['conv', 11, 4, 96], ['max_pool', 3], ['conv', 5, 1, 256], ['max_pool', 3], ['conv', 3, 1, 384], ['conv', 3, 1, 384], ['conv', 3, 1, 384], ['max_pool', 3], ['flatten'], ['dropout', 0.5], ['dense', 1024], ['dropout', 0.5], ['dense', 1024],	256	0.93	0.88	0.88	52.31
100	VGG_16	5	[['VGG16'], ['dense', 512], ['dense', 64], ['output', 10]]	128	0.97	0.84	0.84	49.00
101	VGG_19	5	[['VGG19'], ['dense', 128], ['dense', 32], ['output', 10]]	64	0.80	0.78	0.78	16.00
102	VGG_19	8	[['VGG19'], ['dense', 1024], ['dense', 256], ['output', 10]]	256	1.00	0.90	0.90	512.00