# Tomato disease identification with shallow convolutional neural networks

**Minh Trinh**[1], **Edward Jung**[1], **Linh Trinh**[1]
[1]University High School, Irvine, California

## SUMMARY
**Plant diseases can cause up to 50% crop yield loss for the popular tomato plant. Successful disease management requires precise and timely disease identification methods, but slow and inconsistent diagnoses from human experts or laboratory testing prevents farmers from applying appropriate treatments before the disease takes its toll. As an alternative diagnostic method, a mobile device-based method to identify diseases from photos of symptomatic leaves via computer vision can be more effective due to its convenience and accessibility. Previous research has achieved high accuracy in diagnoses using "deep" convolutional neural networks (CNNs) that have a large number of processing layers and thus a large computational requirement difficult to satisfy on most mobile devices. To enable a practical mobile solution, a "shallow" CNN with few layers, and thus low computational requirement but with high accuracy similar to the deep CNNs is needed. In this work, we explored if such a model was possible. We hypothesized that shallow CNNs would be capable of achieving even higher accuracy than deep CNNs because of shallow CNNs' relatively low inherent dimensionality reduction, allowing it to preserve more distinguishing features between photos of leaves with different diseases. We compared the performance of a shallow CNN we built against conventional deep CNNs. The shallow CNN outperformed several deep CNNs, confirming our hypothesis. This finding enabled us to develop PlantDoctor, a mobile app that can accurately identify plant diseases via leaf images captured by the phone camera in real-time and suggest appropriate treatments.**

## INTRODUCTION
Diseases brought about by insects, pathogens, or lack of nutrition can cause significant crop yield loss for tomatoes, which comprise nearly 20% of all vegetable consumption in the United States (1). For example, the Septoria leaf spot disease, caused by a fungus, can result in 50% reduction of a single tomato crop yield (2). Plant losses from such diseases can cost the global economy 220 billion USD annually (3). Precise and fast identification of diseases afflicting a tomato crop is crucial so the appropriate treatment can be administered as soon as possible to prevent the crop from rotting.

Currently, identification of diseases is done by human experts or laboratory testing methods, such as polymerase chain reaction, immunofluorescence, fluorescence in situ hybridization, and enzyme-linked immunosorbent assay. However, these laboratory methods require expert technicians and can take a lot of time to complete, making fast identification difficult (4). In addition, diagnoses can be inconsistent between different experts (5).

A potential alternative for effective disease diagnosis is a mobile device-based computer vision solution to identify diseases from photos of tomato plants. This method would diagnose diseases in tomato plants by recognizing visible symptoms on leaves, which are often considered reliable indicators of disease (6). A mobile device-based solution would be less expensive, easier to use than laboratory testing, and more precise and consistent than a human expert. Most importantly, its portability in the field enables convenient and fast diagnoses, allowing for timely treatment.

The advancement of computer vision has led to research in using various machine learning convolutional neural networks (CNNs) to identify tomato diseases from photos of tomato leaves. CNNs are a type of machine learning model that take an input image, assign weights and biases to different pixels in the image, and use this to differentiate between different classes, or in this case, diseases (7). CNNs use many different layers, or algorithms, that each uniquely process the image. For example, a convolutional layer uses filters to stride through the image and applies a formula to pixel values within the image (8). These "convolutional filters" can be configured to be larger in size or stacked for more processing, at the cost of higher computational expense. A max pooling layer is one that uses filters to stride through an image and obtain the maximum values of different sections of the image (9). In addition, the architecture of the neural network is modifiable; layers can be swapped, changed, or added as necessary.

There have been several attempts to use CNNs for tomato disease identification in previous studies. Ramakrishna trained a VGG19 CNN (19 hidden layers) model to classify an image of a tomato leaf as healthy or as one of four different diseases. This model achieved 96.0% accuracy (10). Mkonyi, et al. trained a VGG16 CNN (16 hidden layers) to identify the Tuta absoluta pest in tomato plants from images of tomato leaves, attaining 91.9% accuracy (11). Multiple different popular CNN models were compared by Zhang et al. to classify an image of a tomato leaf as healthy or as one of eight diseases. ResNet50 (50 hidden layers) was the most accurate (97.3%), followed by AlexNet (8 hidden layers, 95.8% accuracy) (12).

While this research demonstrates the viability of using CNNs to identify tomato diseases, it would be difficult to use the same VGG16, VGG19, AlexNet, ResNet models mentioned above on mobile devices, as these models are too demanding in terms of computational power and memory space. This is because the CNNs used in the previous research are "deep CNNs"; they have been created with many hidden layers that "inherently apply a form of dimensionality reduction" at great scales, thus requiring computational power difficult for mobile devices to achieve internally (13–15). This limitation of mobile devices can potentially be overcome by using a CNN with a significantly smaller number of layers (a "shallow" network), requiring many times fewer computations without sacrificing accuracy. This inspired our research exploring the effectiveness of "shallow" networks for plant disease identification. As there exists no consensus on the exact boundary between what is considered a "shallow" and "deep" network, for the purposes of this paper, we will consider shallow networks to be anything with four or fewer hidden layers (16). To quantitatively compare resource demand between shallow and deep CNNs, we used floating point operations per second (FLOPS), memory size, duration of training time, duration of CNN loading time, and duration of CNN execution time, all metrics commonly used to quantify resource consumption (15, 17). FLOPS is a measure of computational cost indicated by the number of floating point operations executed per second; a higher number indicates a more resource-intensive computation.

We hypothesized that an optimized shallow CNN would be capable of achieving a higher accuracy than conventional deep CNNs in identification of tomato disease because a shallow network's lower degree of dimensionality reduction may preserve more distinguishing features between plant leaves afflicted by different diseases. To test this hypothesis, we first performed hyperparameter tuning on the shallow network by testing different shallow network architectures to maximize its accuracy before comparison with deep CNNs. These hyperparameters included the number of convolutional layers, number of max pooling layers, number of convolutional filters, and convolutional filter size. Changing these has an impact on computational cost (i.e., required FLOPs, memory size, etc.) with more layers, larger filter sizes, or more filters generally associated with higher costs. After optimizing our shallow CNN, we compared both its accuracy and computational cost with that of conventional deep networks such as VGG16, VGG19, AlexNet, and ResNet. Our experiments produced supporting evidence for our hypothesis; they demonstrated the higher accuracy of our proposed optimized shallow CNN, at 97.1%, outperforming several deep CNNs. This establishes that shallow CNNs are also an accurate method for tomato disease identification using tomato leaf photos. Not only that, but our comparisons also showed that our shallow CNN was considerably lighter in computational cost, requiring at least three times fewer FLOPS, memory, loading time, and execution time than the deep models. This indicates our proposed shallow CNN is substantially more suitable for resource-restricted mobile devices, without loss of accuracy. We then incorporated our CNN into a mobile app, PlantDoctor, to identify plant diseases via leaf images captured by the phone camera and suggest appropriate treatments, with the shallow nature of the CNN allowing for fast, real-time diagnosis without an Internet connection in the field.

## RESULTS

We performed three experiments and one app-building stage. To conduct training and testing of models, we used the PlantVillage dataset, which contains images of tomato leaves labeled as either healthy or infected by one of nine diseases (18). These nine diseases include bacterial spot, early blight, late blight, leaf mold, Septoria leaf spot, two-spotted spider mites, target spot, yellow leaf curl virus, and mosaic virus (**Figure 1**). We split the dataset into a training set (14,623 photos), validation set (2,905 photos), and test set (3,632 photos). Each experiment involved a CNN initialization, training epoch, and testing procedure using the sub-datasets

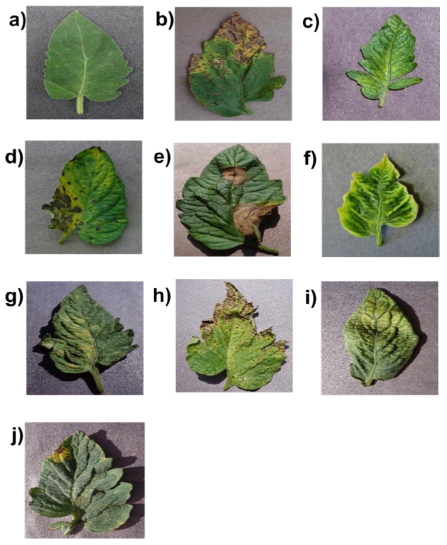

**Figure 1: Tomato dataset disease classes and example photos.** One example photo from each of nine tomato disease classes and one healthy class is shown from the dataset. a) Healthy leaf. b) Bacterial spot. c) Mosaic virus. d) Early blight. e) Late blight. f) Yellow leaf curl virus. g) Leaf mold. h) Septoria leaf spot. i) Spider mites. j) Target spot.

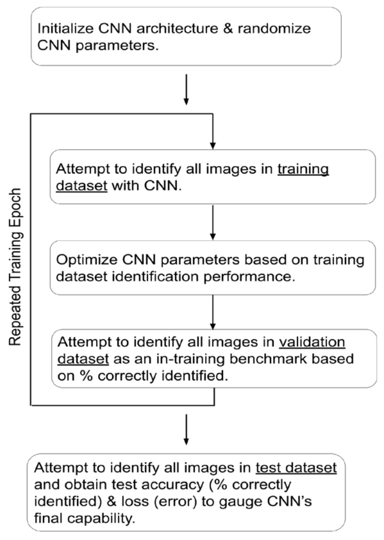
Initialize CNN architecture & randomize CNN parameters.
Attempt to identify all images in training dataset with CNN.
Optimize CNN parameters based on training dataset identification performance.
Attempt to identify all images in validation dataset as an in-training benchmark based on % correctly identified.
Repeated Training Epoch
Attempt to identify all images in test dataset and obtain test accuracy (% correctly identified) & loss (error) to gauge CNN's final capability.

**Figure 2: Training and test procedure for CNNs.** Flowchart showing role of different datasets and the steps to train and test one CNN: initialization, training epochs, and testing. The "Repeated Training Epoch" was repeated a different number of times between experiments depending on the amount of training needed for the CNN.

with various CNN architectures (**Figure 2**). The general structure of our CNN consisted of a series of convolutional and max pooling layers, then one hidden 4096-node "fully-connected" layer for additional computation, a common layer for neural networks (7)

## Stage 1 hyperparameter tuning demonstrates outperformance of shallow CNN with three convolutional layers and three max pooling layers

First, we determined the optimal number of max pooling and convolutional layers in our shallow network. We created nine candidate models, each with M max pooling layers and N convolutional layers where M, N = {1, 2, 3}. Only the convolutional layers and the single fully-connected layer are considered hidden layers, so we only created models with no more than three convolutional layers to keep the total hidden layer count to no more than four (19).

For each of the nine candidate models, we trained them for 65 epochs. In each epoch, we trained the model on the training set, producing a training accuracy, then attempted to identify each image in a separate validation set to test the model's performance on unseen data, producing a validation accuracy. After 65 epochs, we used the model to identify all leaf photos in the test set, recording test accuracy and test loss. Accuracy is the proportion of leaf images in a dataset where the model identified the correct disease, and loss is a measure of the model's error when comparing the model's prediction to the ground-truth label. We used the categorical cross-entropy loss function, which compares the model's predicted probabilities for each class label (10 total: nine diseases and one class for no disease) with the ground-truth class label. The larger the loss, the more incorrect the model was, ranging from predicting the correct class but with low certainty to predicting the incorrect class. Higher accuracy and lower loss indicate better model performance.

We observed that shallow CNNs' accuracies would almost always plateau in training around epochs 55–65 before

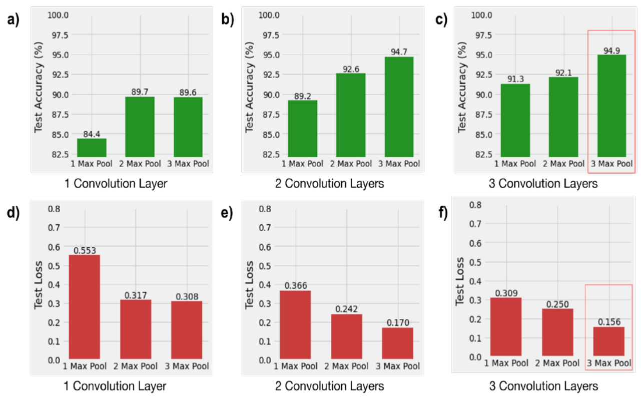

**Figure 3: Stage 1 hyperparameter tuning.** Bar charts comparing the test accuracy and loss of nine candidate CNNs tested in Stage 1 hyperparameter tuning. Trained and tested CNNs with varying convolution and max pooling layers. The red outline highlights the outperformance of a CNN with 3 max pooling layers and 3 convolution layers. Test accuracies for models with 1, 2, or 3 max pooling layers and a) 1, b) 2, and c) 3 convolution layers are shown. Test losses for models with 1, 2, or 3 max pooling layers and d) 1, e) 2, and f) 3 convolution layers are shown.

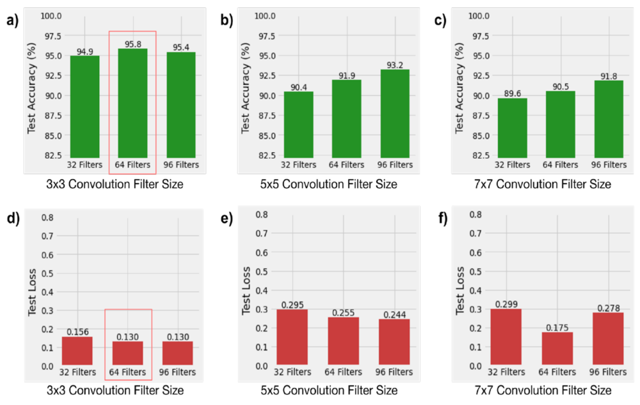

**Figure 4: Stage 2 hyperparameter tuning.** Bar charts comparing the test accuracy and loss of nine candidate CNNs tested in Stage 2 hyperparameter tuning. Trained and tested CNNs with varying convolution filter sizes and quantities. The red outline highlights the outperformance of a CNN with 64 3x3 filters. Test accuracies for models with 32, 64, and 96 filters sized a) 3x3, b) 5x5, and c) 7x7 are shown. Test losses for models with 32, 64, or 96 filters sized d) 3x3, e) 5x5, and f) 7x7 are shown.

declining due to "overfitting," or aligning itself too closely with training data and thus performing inaccurately on the validation or test sets. This was the reason why we trained all shallow CNNs in hyperparameter tuning experiments (Stages 1 and 2) for 65 epochs.

The optimal candidate model from Stage 1 hyperparameter tuning consisted of three convolutional layers and three max pooling layers, with 0.156 test loss and 94.9% test accuracy (**Figure 3**). We kept this model architecture constant in our next stage of hyperparameter tuning.

## Stage 2 hyperparameter tuning demonstrates outperformance of shallow CNN with 64 3x3 convolutional filters

We next determined the optimal number and size of convolutional layer filters. We created nine candidate models, with one model having N convolutional filters each of M size where N = {32, 64, 96} and M = {3x3, 5x5, 7x7}. We repeated the training and test procedure (**Figure 2**). The optimal model from Stage 2 hyperparameter tuning had 64 3x3 convolutional filters, with 0.130 test loss and 95.8% test accuracy (**Figure 4**). With hyperparameter tuning complete, we had determined our optimal shallow CNN architecture (**Figure 5**).

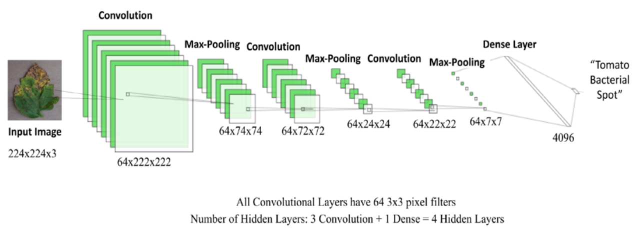

**Figure 5: Optimized shallow CNN architecture.** CNN diagram showing the setup of each layer in our optimized shallow CNN and visualizing the processing done on an example image. This architecture was determined through the hyperparameter tuning procedure.

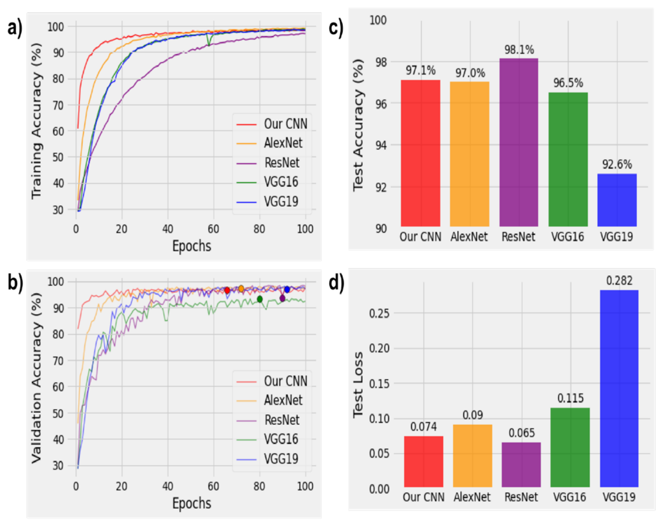

**Figure 6: Comparison of accuracy and loss performance between optimized shallow CNN and conventional deep CNNs.** All CNNs were trained for 100 epochs, restored to the epoch it achieved highest validation accuracy ("Best Version"), then tested on the test set to produce test accuracy and loss. A) Training accuracies on training dataset over 100 epochs for all CNNs. B) Validation accuracies on the validation dataset over 100 epochs for all five CNNs, and colored points showing the epoch of the Best Version (our CNN: 65 epochs, AlexNet: 71 epochs, ResNet: 89 epochs, VGG16: 79 epochs, VGG19: 91 epochs). C) Bar chart comparing test accuracies among all five CNNs. D) Bar chart comparing test losses among all five CNNs, with our shallow CNN trailing only ResNet. Lower test loss indicates better test accuracy.

## Comparison of optimized shallow CNN and deep CNNs demonstrates outperformance of shallow CNN for all deep models except ResNet

We investigated if our optimal shallow CNN was more accurate than conventional deep CNNs. We recreated the four deep CNN models VGG16, VGG19, AlexNet, and ResNet following their documented architectures. We repeated the training and testing procedure on our final optimized shallow CNN and all four deep CNNs. However, this time we graphed training accuracy over 100 training epochs instead of 65, as the deep CNNs can require more training than shallow CNNs (**Figure 6A**). However, increasing the number of epochs made the modes prone to overfitting. Thus, we implemented a procedure to restore the best version of each model (when it achieved highest validation accuracy) from the training period. The best versions were run on the test dataset to produce a test accuracy and loss. We also timed how long it took to train each CNN for 100 epochs. Our optimized CNN produced the second-highest accuracy and second-lowest loss at 97.1% and 0.074 loss, underperforming ResNet (98.1% accuracy, 0.065 loss), but outperforming three deep models, AlexNet, VGG16, and VGG19 (**Figure 6C-D**). In addition, our CNN trained to its best version at 65 epochs, before AlexNet at 71 epochs, ResNet at 89 epochs, VGG16 at 79 epochs, and VGG19 at 91 epochs (**Figure 6B**).

After we finished training our shallow CNN and deep CNNs, we collected data on various factors related to the CNNs' training and execution. This data helped us compare the performances of the shallow and deep CNNs and illustrate the advantage of using shallow CNNs. We looked at

| | Parameters | Train Time (min) | Required Operations (Gigaflops) | Load Time (sec) | Execution Time (sec) | Memory (Megabytes) |
|---|---|---|---|---|---|---|
| Our shallow CNN | 6.5M | 47 | 0.63 | 2.3 | 0.7 | 25 |
| AlexNet | 62.3M | 72 | 2.01 | 6.4 | 4.1 | 233 |
| ResNet | 23.6M | 232 | 7.75 | 6.7 | 7.3 | 98 |
| VGG16 | 138.4M | 275 | 31.00 | 13.8 | 9.6 | 528 |
| VGG19 | 143.7M | 318 | 39.30 | 15.7 | 11.6 | 548 |

**Table 1: Computational cost comparison of shallow and deep learning CNNs.** Statistics for our shallow CNN and conventional deep CNNs illustrating the shallow CNN's lightness in memory and computational usage, and speed in loading and execution compared to the deep CNNs. The CNN number of trainable parameters is also included.

the CNNs' properties to determine the parameter counts and obtained training time as mentioned above. We measured the time for each model to load and run inference on a single image (load and execution time) and estimated the number of FLOPS required for each. Our shallow CNN is the most resource-efficient based on these factors: 47 minutes to train, using 25 megabytes (MB), requiring 0.63 gigaflops of operation, using 2.3 seconds to load, and taking 0.7 seconds to run inference. AlexNet (2.01 gigaflops and 223 MB) and ResNet (7.75 gigaflops and 98 MB) were less efficient. VGG16 and VGG19 showed markedly higher resource consumption, both 30+ gigaflops and 500+ MB (**Table 1**).

## Construction of the mobile application and extension of functionality to other plants

After construction and optimization of the shallow CNN, we created a mobile app named PlantDoctor to perform inference on uploaded images or in real-time using the CNN's weights that we downloaded onto a smartphone (**Figure 7**). We also expanded the app's functionality to eight other plants – corn, grape, potato, apple, cherry, peach, bell pepper, strawberry – by creating new CNNs with our optimized shallow architecture and retraining each to diagnose diseases for a different plant, using the other plants' disease-labeled photos from the PlantVillage dataset for training. These models all achieved greater than 95% test accuracy (**Figure 7**). The total storage size of all nine models on the app totaled around 225 MB, still smaller than a single AlexNet, VGG16, or VGG19 made to diagnose diseases for one plant.

## DISCUSSION

We confirmed our original hypothesis that a shallow CNN achieves higher accuracy than conventional deep CNN methods in identifying tomato diseases while being more computationally efficient. Our optimized shallow CNN to identify tomato diseases resulted in a higher test accuracy and lower test loss than three of four deep CNNs tested. Our optimized shallow CNN achieved a high accuracy of 97.1% and low loss of 0.074, underperforming only one of four deep models: ResNet at 98.1% accuracy, 0.065 loss (**Figure 6C-D**).

However, the speed and lightness of our CNN make it much more practical for mobile apps than the ResNet, outweighing the marginal difference in accuracy. In terms of required operations to run inference, our CNN, at 0.63 Gigaflops, is over 10 times more efficient than ResNet at 7.75 Gigaflops (**Table 1**). In addition, our CNN has a loading time more than three times less and execution time more than ten times less

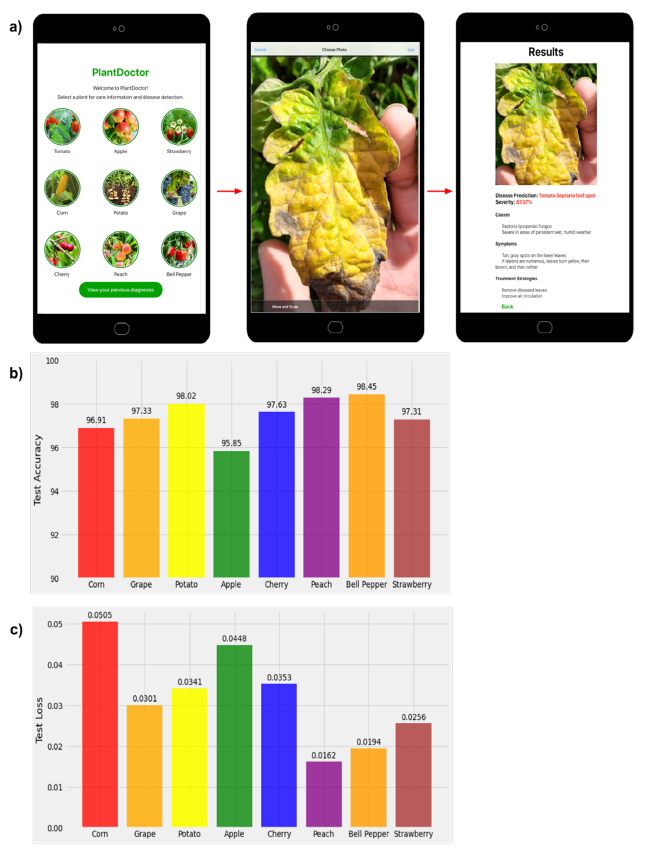

**Figure 7: PlantDoctor mobile app and performance of CNNs made to diagnose other plants.** A) Screenshots showing the app's usage. Users can select a type of plant to diagnose, point the phone camera to a diseased leaf of the plant, and obtain a diagnosis with information on the disease and appropriate treatments. B) Test accuracies and C) test loss values of CNNs trained to diagnose other plants are shown.

than ResNet, allowing for a much better user experience and even real-time usage (**Table 1**). Finally, with the shallow CNN having nearly four times lower memory usage than ResNet, the shallow CNN makes for a much lighter mobile app (**Table 1**).

The training time of models is not as crucial to consider with regard to mobile development; models' weights can just be extracted and used for inference on a mobile device after they are trained on other machines. However, it is still interesting to note that our shallow CNN took less time than other models to train. ResNet took around four hours to train to achieve its marginally better accuracy, while the shallow CNN took around one hour (**Table 1**). This is because more layers in a deep CNN like ResNet means there are more parameters that need to be optimized through the training process. ResNet had over 20 million parameters, while our shallow CNN had around 6 million, resulting in substantially faster convergence and lower training time (**Table 1**). This can be visualized in the training graphs; the shallow CNN's training and validation accuracy curve (red) during training is much steeper than that of any deep model, and it achieves its highest validation accuracy earliest (**Figure 6A-B**).

Despite mostly converging in the 90–100% validation accuracy range after 100 epochs, all the models had widely different starting accuracies, with ResNet, VGG16, and VGG19 starting near 30% while AlexNet and our CNN started higher (**Figure 6B**). We believe this is also a function of parameter counts; CNN parameters are initialized in a pseudorandom manner, so at first, a larger model like VGG16 at 138.4 million parameters may effectively be making more random and inaccurate classifications on the validation set than AlexNet at 62.3 million parameters (**Table 1**).

The above analysis shows that our shallow CNN is substantially more suitable for mobile development due to its low resource consumption and greater accuracy than some deep learning models. This allowed us to develop PlantDoctor, a mobile application for farmers and growers that utilizes CNNs to identify plant diseases via leaf images captured by the phone camera and suggest appropriate treatments (**Figure 7**). The app functions as a mobile expert system and is an effective alternative to diagnosing diseases with human experts and other laboratory methods. The low execution time of the shallow CNN allows interpretation of leaf images in real time; users only need to point their phone camera at a plant leaf to receive a near-instant diagnosis. Furthermore, due to the model's small size, we have been able to expand the number of plants that the app can diagnose to eight other plants by storing a CNN for each plant all simultaneously within the app. The CNNs trained to diagnose diseases for other plants achieved similar high accuracies to our original tomato disease CNN. If instead deep CNN architectures were trained to diagnose diseases of all nine plants and stored on the app, the app would have been significantly larger in size and require more time to load each model. As the CNNs are easily stored completely within the device, the app does not require an Internet connection to run and can be taken out into remote fields. In the future, we will be able to expand to many more plants while keeping the app small.

We were subject to some time and resource constraints; thus, we had to limit hyperparameter tuning to optimizing the most influential hyperparameters. In the end, this limited optimization seemed be sufficient, as our optimized shallow CNN could outperform deep CNNs.

In the future, we can explore the application of our shallow CNN to predict diseases by training with datasets of leaves with smaller or fewer symptoms, allowing growers to take preventive measures against diseases in their early stages before the infection becomes a larger problem. We can also explore the identification of diseases through photos of symptoms on other plant parts like the fruit or stem, which is useful when symptoms are absent on leaves. When expanding the functionality of our CNNs, we can use transfer learning in our training process. Transfer learning is an approach where models that have already been trained are reused as the starting point for training on a different data set. The relatively fast training time of our shallow CNN will help this process be efficient.

Our research demonstrates that shallow CNNs are a feasible and accurate approach to identifying plant diseases, without needing the large amounts of computational resources, execution time, and memory space required by conventional deep CNNs. By utilizing our app, farmers can obtain fast, reliable plant diagnosis of diseases, allowing them to take earlier action, reducing the financial loss of unsalvageable plants. With our CNN's successful integration into a mobile application, we believe we have created a tool helpful for farmers and growers to get precise and immediate identification of plant diseases allowing for timely and appropriate treatments.

## MATERIALS AND METHODS

We used Google Colaboratory, an online Python notebook with Cloud GPU, as the platform to do all our experiments. We used TensorFlow, a Python library for machine learning model development, to create, train, and test CNNs. All CNNs we created in our experiments were trained with categorical cross-entropy loss, Rectified Linear Unit (ReLU) activation for hidden layers, and the stochastic gradient descent optimization function.

### Data preparation

We obtained 18,160 disease-labeled tomato leaf images from the PlantVillage Dataset. We shuffled the dataset and split it into 64% for the training set, 16% for the validation set, and 20% for the test set. Images were resized to 224 x 224 pixels to fit the default image size for the deep CNNs trained later on.

To augment the training dataset, in other words to provide CNNs with more data to learn the patterns of each disease during training, we applied color changes, rotations, and flips to 25% randomly-chosen images from the training dataset. We further augmented the training data by adding 3,000 images of labeled tomato plant leaves whose backgrounds were replaced with a solid color, unlike the original 18,160. This increased the number of images in the training dataset from 11,623 (64% of the original 18,160) to 14,623.

The Python libraries NumPy, pandas, and TensorFlow were used for data preparation procedures.

### Hyperparameter tuning Stage 1

Nine CNNs were created with N convolutional layers, M max pooling layers, one 4,096-node fully-connected layer, and an output layer with softmax activation, where N and M $\epsilon$ {1, 2, 3}. These were trained for 65 epochs each. The CNN with highest test accuracy and lowest test loss determined the optimal number of convolutional and max pooling layers in our shallow network.

### Hyperparameter tuning Stage 2

We created nine CNNs with three convolutional layers with N convolutional filters each of M size, three max pooling layers, one fully-connected layer, and one output layer, where N $\epsilon$ {32, 64, 96} and M $\epsilon$ {3x3, 5x5, 7x7}. These were trained for 65 epochs each. The CNN with highest test accuracy and lowest test loss determined the optimal number and size of convolutional filters for the shallow network. We constructed a CNN with optimal hyperparameters determined from Stage 1 and Stage 2 hyperparameters, called the "optimized shallow CNN."

### Comparison of shallow and deep networks

We recreated the four deep CNN models VGG16, VGG19, AlexNet, and ResNet following their documented architectures (20–22). Each of the four deep CNNs and our optimized shallow CNN was trained for a maximum of 100 epochs and then tested. We accounted for potential overfitting by restoring the best version of the model that achieved the highest validation accuracy during training. This was done by implementing a callback to re-save the model's weights during training every time it achieved a new high in validation accuracy. The final trained model was tested on the test dataset to produce a test accuracy and test loss. The test accuracy and test loss of the shallow CNN was compared to those of the other deep CNNs' to determine if it was capable of outperforming other deep CNNs.

To compare the computational costs between our shallow model and the deep models, we collected parameter counts, using the *TensorFlow* summary function; gigaflops of operation using the analytical library *keras-flops*; and training time, using a timer to record the duration of the 100 epochs of training. We then transferred the models' weights to a mobile app and timed how long it took to load and execute the models.

### Building the mobile app

We built the user interface of the mobile app with React Native (23). We downloaded the shallow CNN's weights and transferred them to the mobile app's storage. The Tensorflow.js (24) library was used to load the CNNs and run inference on photos taken in real-time frames or from the camera roll.

To expand the app's functionality to more plants, we downloaded data from the PlantVillage dataset for eight other plants: corn, grape, potato, apple, cherry, peach, bell pepper, strawberry. We made new copies of the shallow CNN architecture with randomized initial weights. We trained one copy on 80% of each of the plants' datasets and tested them on the remaining 20% to find the test accuracy and loss. When finished, we downloaded the other plant CNN models' weights to the mobile app and set up TensorFlow.js to run the CNN requested by the user.

## ACKNOWLEDGEMENTS

Our thanks go out to Dr. Ash Pahwa, a professor at California Institute of Technology from whom we acquired substantial machine learning knowledge, Elvin Kang who helped us with proofreading and gave us pointers for using TensorFlow for Machine Learning, and Jasmine Delgado, who gave us helpful guidance in creation of the mobile app. In addition, we would like to thank Mr. Nathan Gipple (Director of Farm and Food Labs) and the network of Orange County master gardeners for providing us with information about tomato plants and diseases as well as advice regarding our mobile application and project.

**Received:** July 19, 2022
**Accepted:** September 21, 2022
**Published:** March 3, 2023

## REFERENCES

1. Reimers, Kristin J., and Debra R. Keast. "Tomato Consumption in the United States and Its Relationship to the US Department of Agriculture Food Pattern." *Nutrition Today*, vol. 51, no. 4, 2016, pp. 198–205., doi:10.1097/nt.0000000000000152.
2. Panno, Stefano, et al. "A Review of the Most Common and Economically Important Diseases That Undermine the Cultivation of Tomato Crop in the Mediterranean Basin." *Agronomy*, vol. 11, no. 11, 2021, p. 2188., doi:10.3390/agronomy11112188.
3. He, Sylvia, and Kate M. Creasey Krainer. "Pandemics of People and Plants: Which Is the Greater Threat to Food

Security?” *Molecular Plant*, vol. 13, no. 7, 17 June 2020, pp. 933–934., doi:10.1016/j.molp.2020.06.007.
4. Fang, Yi, and Ramaraja P. Ramasamy. “Current and Prospective Methods for Plant Disease Detection.” *Biosensors*, MDPI, 6 Aug. 2015, www.ncbi.nlm.nih.gov/pmc/articles/PMC4600171/.
5. Mahlein, Anne-Katrin. “Plant Disease Detection by Imaging Sensors – Parallels and Specific Demands for Precision Agriculture and Plant Phenotyping.” *Plant Disease*, vol. 100, no. 2, 2016, pp. 241–251., doi:10.1094/pdis-03-15-0340-fe.
6. UCANR. “How to Manage Pests.” *UC IPM Online*, 2020, ipm.ucanr.edu/home-and-landscape/tomato/index.html.
7. Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks-the eli5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
8. IBM Cloud Education. “What Are Convolutional Neural Networks?” IBM, 20 Oct. 2020, www.ibm.com/cloud/learn/convolutional-neural-networks.
9. Alzubaidi, Laith, et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.” *Journal of Big Data*, vol. 8, no. 1, 2021, doi:10.1186/s40537-021-00444-8.
10. Ramakrishna, Rajath. “Machine Learning Based Approach in Detection and Classification of Tomato Plant Leaf Diseases.” *NORMA EResearch*, 11 June 2020, norma.ncirl.ie/4275/1/rajathramakrishna.pdf.
11. Mkonyi, Lilian, et al. “Early Identification of Tuta Absoluta in Tomato Plants Using Deep Learning.” *Scientific African*, vol. 10, Nov. 2020, doi:10.1016/j.sciaf.2020.e00590.
12. Zhang, Keke, et al. “Can Deep Learning Identify Tomato Leaf Disease?” *Advances in Multimedia*, vol. 2018, 2018, pp. 1–10., doi:10.1155/2018/6710865.
13. Kim, Daniel E. “Comparison of Shallow and Deep Neural Networks in Network Intrustion Detection.” *California State University*, 28 Nov. 2017, scholarworks.calstate.edu/downloads/fj236307r.
14. P, Karthika. Re: “Do You Think That Convolutional Neural Network Is Good for Mobile Application?” *ResearchGate*, 25 July 2019, www.researchgate.net/post/Do_you_think_that_Convolutional_Neural_Network_is_good_for_mobile_application.
15. Ogden, Samuel, and Tian Guo. “Characterizing the Deep Neural Networks Inference Performance of Mobile Applications” *ArXiv*, 10 Sept. 2019, arxiv.org/pdf/1909.04783.pdf.
16. Lei, Fangyuan, et al. “Shallow Convolutional Neural Network for Image Classification.” *SN Applied Sciences*, vol. 2, no. 1, 2019, doi:10.1007/s42452-019-1903-4.
17. Lu, Zongqing, et al. “Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices.” *Proceedings of the 25th ACM International Conference on Multimedia*, 2017, doi:10.1145/3123266.3123389.
18. Mohanty, Sharada. “SpMohanty/PlantVillage-Dataset: Dataset of Diseased Plant Leaf Images and Corresponding Labels.” *GitHub*, GitHub, 23 Sept. 2018, github.com/spMohanty/PlantVillage-Dataset.
19. Dhami, Dharti. “Convolutional Neural Networks: Layers.” *Medium*, Medium, 21 Dec. 2018, medium.com/@dhartidhami/convolutional-neural-networks-layers-3bc2c9121678.
20. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv* preprint arXiv:1409.1556 (2014).
21. Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” *Communications of the ACM*, vol. 60, no. 6, 2017, pp. 84–90., https://doi.org/10.1145/3065386.
22. He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” 2016 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, https://doi.org/10.1109/cvpr.2016.90.
23. “React Native · Learn Once, Write Anywhere.” *React Native RSS*, Meta Platforms, Inc., 2023, https://reactnative.dev/.
24. “TensorFlow.js: Machine Learning for JavaScript Developers.” *TensorFlow*, 2023, https://www.tensorflow.org/js.