Final Report: Plant Disease Detection using CNNs

Team 2: Yuqing Zhao 10175301, Kasha Talaga 10170688, Reza Jafari 30123125, Evan Boerchers 10172639, Hunter Kimmett 10096647

Link to our GitHub repository: https://github.com/thelisazhao/ENEL645_PlantDiseaseDetection

1. Introduction

1.1. Problem

There are three crop species in the dataset: tomato, bell-pepper and potato. Our team is classifying the diseases in an individual model for each crop species. The potato species contains three classes: healthy, late blight and early blight. The bell-pepper species contains two classes: healthy and bacterial spot. The tomato species contains four classes: healthy, leaf mold, mosaic virus, and target spot.

Our models for the potato and bell pepper were summarized in the midterm report. In this final report a model for the tomato dataset will be included, as well as outlining any improvements we have made to the other models. The team will not be creating a model to classify multiple species and diseases together. Novel aspects include using tensorflow and keras to develop deep learning models and applying our model to a different dataset to test generalizability.

1.2. Motivation

Modern technologies have given human society the ability to produce enough food to meet the demand of more than 7 billion people. However, food security and producing healthy food are essential. Food security still remains challenging and it is threatened by a number of factors including climate change, the decline in pollinators, plant diseases, and others [1]. Plant diseases are not only a threat to food security at the global scale, but can also have disastrous consequences for smallholder farmers whose livelihoods depend on healthy crops.

Identifying crop diseases correctly is the first step to disease management. Proper diagnoses are important to prevent disease spread. Deep learning plant health detection will save time and money for food growers. Recent development into smartphones has allowed high quality images to be taken by a user. Along with the improved computational complexity of smartphones, disease detection based on image detection is feasible. Utilizing this opportunity to reduce crop wastage will ensure better food security for the future [1].

2. Materials and Methods

In this project the team will utilize existing datasets to develop a plant health detection algorithm using deep learning. The deep learning architecture will be TensorFlow, written with Python code, while the data will be displayed with Keras.

2.1. Dataset

The team will use the public Plant Village dataset available on Kaggle [2]. The Plant Village dataset is promising because there are over 54,000 images with multiple crop species and diseases. Training a neural network requires time and, most importantly, a large dataset. In this dataset the class labels are provided as a crop-disease pair.

Our dataset will contain a subset of crop species from the Plant Village dataset. The team will use tomato, potato and bell pepper species. The tomato images have 4 disease classes, the bell pepper images have 2 disease classes, and the potato images have 3 disease classes. An additional dataset of images found on the internet will be used to test generalizability; we will refer to this as the generalized dataset. As an example, Figure 1 presents potato images from both datasets. For each individual crop species (tomato, potato, or bell pepper) the team will create a model to classify the related crop diseases.

The results for the potato and bell pepper model were discussed in the midterm report. In this report, model improvements will be outlined and the tomato model results will be provided. The tomato model was trained using the TALC cluster, as this dataset contained a large amount of images.

2.2. Approach

The database being used includes image class labels. As prior class knowledge will be an input to the classification algorithm our design involves a supervised learning approach. The image data will be split into train, validation and test sets. The selection of these images will be random to ensure that all groups are represented in the data.

The split-folders package in python was used to create a development and test folder with images from all classes. The development and test folder contain 80% and 20% of the total images respectively. Within the folders there are 9 sub-folders with names corresponding to each class label. The development set can be further split into train and validation sets using ImageDataGenerator. Depending on the model a train/validation split of 80/20 or 90/10 was used. To use a particular species of plant, the associated classes can be specified in flow_from_directory.

The supervised algorithm will analyze the input training data to determine class labels of unseen data. Our training mechanism will be from scratch. In addition to the allocated test set, the team will use a generalized test dataset (not from the PlantVillage dataset) for testing the model. This serves to test the generalizability of our model.

2.3. Data Pre-processing

To define our preprocessing pipeline the tensorflow keras ImageDataGenerator was implemented. Using this python object allowed for preprocessing parameters to be specified and applied to loaded data. This processing includes normalization, train/validation split, and data augmentation, where the augmentations specified in the parameters will be randomly applied to generated images.

Given the fact that the data is currently divided into development and test directories, the most efficient method of data loading is to use the tensorflow keras flow_from_directory. This is important because images will not be loaded into the clusters RAM, saving memory. The flow from directory function will handle image resizing, shuffling, label encoding as well as specifies batch size. The flow from directory function is a function of ImageDataGenerator, so to apply the defined preprocessing pipeline of the ImageDataGenerator the flow_from_directory function was called, and given the desired directory.

As a result of the aforementioned steps, the data flow defined using ImageDataGenerator can seamlessly be fed into the model, allowing the keras library to handle the loading of resources, optimizing memory usage.

2.4. Model Summary

For this project a deep learning approach with a convolutional neural network (CNN) will be implemented. This network will contain convolutional layers, pooling layers, activation layers (ReLU and softmax), and fully connected layers. The team will define our own architecture as opposed to using a predefined one. The CNN uses nominal processes to detect and categorize images [2]. As an example, Figure 2 shows the summary of model parameters for the potato model, which is similar to both the bell pepper and tomato models.

2.4.1. Potato Model

The potato model contains three classes. Figure 2 shows the summary of the model parameters. Inspired by the general architecture of VGG model, which has an easy to understand modular structure [3]. The model stacks convolutional layers with a batch normalization, max pooling and dropout layer. Those blocks have 32, 64, and 128 filters which increase the depth of the network. The introduction of batch normalization was an attempt to accelerate the learning process [4]. The regularization technique, dropout, is slowly increased to offset the acceleration of batch normalization. To help with generalization, the validation and training set was split by 10% and 90% respectively.

2.4.2. Bell Pepper Model

For the classification of the bell-pepper data which contains two classes, healthy and bacterial spots, a CNN model is used. The summary of the model and the number of parameters are similar to the potato model shown in Figure 2. As mentioned in subsection 2.4.1, a structure similar to the VGG model was used. The model stacks convolutional layers with a batch normalization, max pooling and dropout layer. These blocks have the same filter sizes as the potato model. Weight regularization techniques were used to improve model generalizability.

2.4.3. Tomato Model

There are 4 classes in the tomato model to be classified. As in the previous models, a CNN architecture inspired by VGG has been implemented. The model stacks convolutional layers with batch normalization, max pooling and gradual dropout layers. These blocks have 32, 64, and 96 filters. Weight regularization techniques and changing the training/validation split from 80/20 to 90/10 were applied to improve model generalizability.

2.5. Model Generalization

Generalization testing was employed for the potato, tomato and bell pepper models. The generalized datasets consisted of images hand-picked from Google images, as shown in Figure 1. The amount of generalized images found for a class of each model ranged from 10-30 images. The team had a difficult time finding clear singular leaf images so the quantity for each disease differed. Upon testing these datasets, the test accuracy was quite low for all models. Given that the bell pepper model has the lowest number of classes (i.e., two), it was expected to have a high test accuracy compared to the other models.

In our initial models, the low generalizability was the result of the model overfitting to the training dataset. Since the training data was very homogeneous, the models needed to be trained to preserve high validation accuracy (95% plus) while not overfitting to these images, allowing for generally good results when running predictions on images outside the Plantvillage data.

In order to improve generalizability of our models the following methods were tested. Different combinations of methods were tested on each model with the best result being selected for the final model.

- Lowering patience so model training stops earlier
- Adding and adjusting dropout layers
- Using batch normalization
- Weight regularization techniques
- Modifying the train/validation split size
- Modifying data augmentation parameters (i.e., changing fill settings)
- Self-supervised learning (attempted for the potato and tomato models)

2.6. Metrics to Assess the Results

In a similar deep learning system [1] the F1 score was used to compare results of various model configurations during training. F1 score is the harmonic mean of the precision and recall of a classification. In class, we have mainly used accuracy as the metric to assess the performance of our model. Below are the equations for the metrics used in this project [5]:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + False\ Positives + True\ Negatives + False\ Positives} \quad \text{(Eq. 1)}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad \text{(Eq. 2)}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad \text{(Eq. 3)}$$

$$F1 = \frac{2*Recall*Precision}{Recall + Precision} \quad \text{(Eq. 4)}$$

These metrics are used to evaluate the effects of different experimental parameters such as the train-test split, data set type, preprocessing steps and training mechanism. To properly measure the success of our model, metrics will be used to evaluate performance on data from the Kaggle dataset and our generalized test set. The metrics will also be used to compare our model to random classification.

To measure the loss and evaluate the fit of our model, the default metric of Cross-Entropy Loss was used for our initial results. We will also be implementing a custom Macro Double Soft F1 Loss function to improve the results of our F1 Scores [6]. The metric implemented to evaluate our model's success is accuracy, however we will also use F1 Score as a metric to compare our model to the generalizations done in [1]**.**

Initially, our team planned on also using Recall and Precision in different loss functions to make the model practical for real life applications. This proved to be impossible for us to accomplish, as using these metrics for a single class in the loss function is not something we were able to code. As far as our research indicates, in practice, it is not used.

3. Results and Discussion

3.1. Potato Results

The potato model has a very high train and validation accuracy of 98.2 and 99.0 percent respectively. The test accuracy is also 99.0 percent. This high accuracy could be a result of our small dataset size as there are only 2000 potato images from the three classes. The results of the train and validation sets after

15 epochs converge and continue to converge to the end of 100 epochs. Early stopping is implemented to prevent overfitting.

The model currently has a generalized dataset of 31 images belonging to 3 classes. Testing the model with this dataset resulted in a test accuracy of 48-55%. This could be due to the size of the dataset not being large enough to create a generalizable model. It can also be due to the number of classes in the dataset. As the classes increase, it becomes harder to generalize. One way to overcome this was to change the validation and training dataset split to 10% and 90% respectively which increased generalizability. In addition, fill mode was set to nearest to better show the image.

The team attempted self-supervised learning (SSL) to improve generalizability of the model. SSL trains a network on a pretext task (image rotation in this case), then the knowledge is transferred to a downstream task (the potato disease classification). Since a large amount of images needed to be loaded into Collabatory, TALC was utilized to handle the large amount of images. This resulted in TALC memory errors when running pretext tasks. This made it impossible for us to do SSL on the potato model with the time we had left.

Interestingly, the initial 15 epochs showed unusual behaviour as the train and validation loss graphs fluctuate. One way to reduce this behaviour is to increase the batch size to 32, reducing the overall noise and fluctuations. The behaviour shown in Figure 3 is not concerning because the training and validation losses decrease as the accuracies increase.

3.2. Bell Pepper Results

The bell pepper classification model runs for 100 epochs and the selected batch size for the model is 32. After 100 epochs, the train accuracy, validation accuracy, and test accuracy are 99.8, 97.5, and 97.4 percent respectively, as shown in Figure 4. The high accuracy is because of the small bell pepper dataset, containing only two classes. This may cause accuracy to decrease when testing with new images not from the PlantVillage dataset for validation.

When testing the model with our generalized dataset we were able to achieve a test accuracy as high as 78.6%. To achieve this, the early stop patience was lowered to 15, dropout was added after each layer block, and weight regularization was implemented using a kernel regularizer with l1 and l2 regularization. Considering how nonuniform the images in the generalized dataset are, this accuracy is very good, and achieving any higher may not be possible given the training data.

3.3. Tomato Results

The results of the tomato model give a high train, validation and test accuracy of 99.2, 98.6, and 98.9 percent respectively. There are fluctuations while training, however, after about 40 epochs the train and validation sets converge and continue to converge to the end of 100 epochs. This is shown in Figure 5. The model implements early stopping to prevent overfitting.

The generalized dataset was used to test our model and the resulting test accuracy was 25-32%, which is quite low. Given that the tomato dataset has 4 classes that are very similar may be the reason for such a low test accuracy. Also, the tomato generalized dataset only contains approximately 20 images. To increase the generalized test accuracy the train/validation split was changed to 90/10, dropout was used, the fill mode was set to nearest, and a similar weight regularization technique as discussed in 3.2 was implemented. Even with these changes the test accuracy did not significantly improve.

As for the potato model, the team attempted to implement a SSL approach, which trains a network on a pretext task (image rotation in this case), then the knowledge is transferred to a downstream task (the tomato disease classification). The team needed to use TALC for the large amount of images, however, an out of memory error appeared when training the pretext task. In the future, our team could further investigate this issue.

3.4. TALC Usage

For this project, we were given access to the University of Calgary's Teaching and Learning Cluster (TALC) in order to speed up the training of our models. We found some moderate success using this service, however the results slightly differed from the results we achieved using Google Colaboratory. The speed at which the models trained was not much faster than Colaboratory, and almost universally delivered slightly worse results for each model. This could have been a random chance as we did not have the time to train many different models to see if these results were consistently lower.

The main use of the TALC came with training and testing models using the F1 Loss function. We had completed our best models for all 3 plants before being able to implement the TALC, so it was used for making experimental modifications. After an initial steep learning curve, we were able to easily implement the TALC to run these experimental models. Unfortunately, one of the downsides of TALC was a lack of interactivity, as scripts cannot be in IPython format. This meant we were unable to use Tensorboard to evaluate the fit of our experimental models.

Tables 1.A and 1.B show the results of these TALC experiments, with Table 1.A showing our baseline results from our standard models, and Table 1.B using the F1 Loss function. Using F1 as the loss metric produced unexpected results and did not improve F1 scores, which could have been random chance but also this may indicate the function is unsuitable for this use. Our F1 Scores for our standard model in the PlantVillage Dataset were in line with and comparable to the scores achieved by S. Mohanty et al. [1].

4. Conclusion

Food scarcity will continue to increase as the earth's resources become more scarce. The best way to improve the quality of foods is to reduce wastage. Using deep learning is one way to automate detection of crop diseases before it is too late. The models that the team have created provide a foundation for applications to upload photographs of potato, tomato and bell-pepper plants in order to detect diseases. The implementation of generalizable models allows even better detection of diseases from a certain species of plant. The team believes that usage of large clusters such as TALC will further allow machine learning algorithms to better train models. Future work may include expanding the dataset or working with local farmers in order to collect better data to create more generalizable models.

5. References

[1] S. Mohanty, D. Hughes, and M. Salathe, "Using Deep Learning for Image-Based Plant Disease Detection," 1Digital Epidemiology Lab, EPFL, Switzerland, p. 7, [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1604/1604.03169.pdf.

[2] "PlantVillage Dataset." https://kaggle.com/emmarex/plantdisease (accessed Feb. 03, 2021).

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ArXiv14091556 Cs*, Apr. 2015, Accessed: Mar. 08, 2021. [Online]. Available: http://arxiv.org/abs/1409.1556.

[4] J. Brownlee, "How to Accelerate Learning of Deep Neural Networks With Batch Normalization,"

Machine Learning Mastery, Jan. 17, 2019. https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/ (accessed Mar. 08, 2021).

[5] A. C. Muller and S. Guido, *Introduction to Machine Learning with Python [Book]*. O'Reilly media, 2018.

[6] "The Unknown Benefits of using a Soft-F1 Loss in Classification Systems," *Medium*. https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d (accessed Apr. 06, 2021).

6. Figures and Tables

(a) (b) (c)

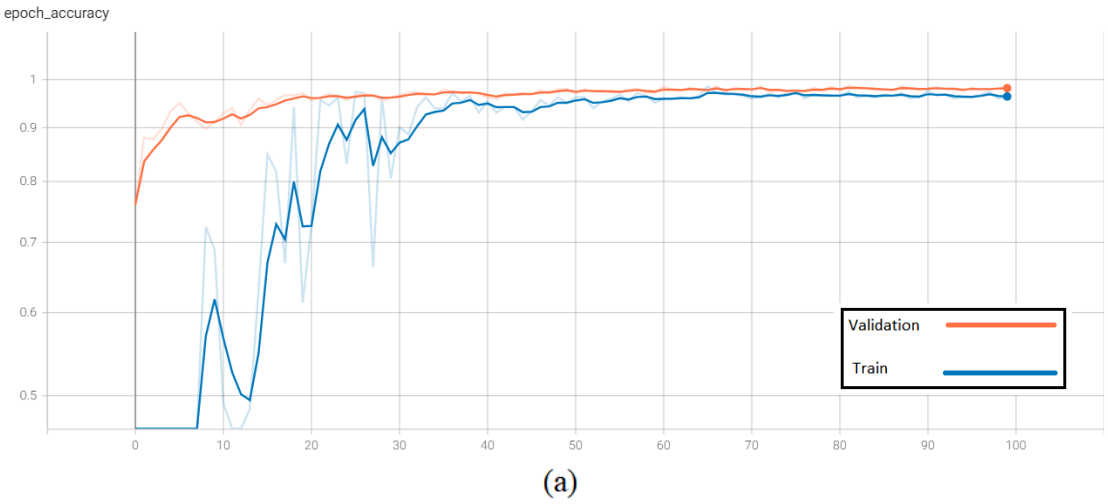
(d) (e) (f)

Figure 1: Example of potato images from the PlantVillage dataset [2] and Generalized potato dataset. (a) PlantVillage healthy potato, (b) PlantVillage potato early blight, (c) PlantVillage potato late blight, (d) Generalized dataset healthy potato, (e) Generalized dataset potato early blight, (f) Generalized dataset potato late blight.

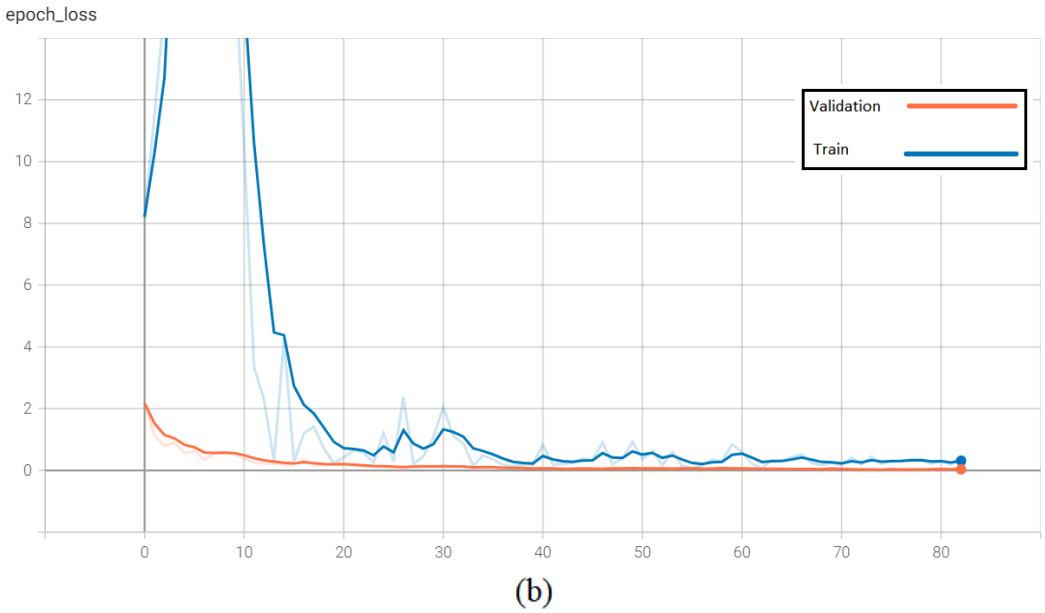
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 250, 250, 3)]	0
conv2d (Conv2D)	(None, 250, 250, 32)	896
conv2d_1 (Conv2D)	(None, 250, 250, 32)	9248
batch_normalization (BatchNo	(None, 250, 250, 32)	128
max_pooling2d (MaxPooling2D)	(None, 125, 125, 32)	0
dropout (Dropout)	(None, 125, 125, 32)	0
conv2d_2 (Conv2D)	(None, 125, 125, 64)	18496
conv2d_3 (Conv2D)	(None, 125, 125, 64)	36928
batch_normalization_1 (Batch	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 62, 62, 64)	0
dropout_1 (Dropout)	(None, 62, 62, 64)	0
conv2d_4 (Conv2D)	(None, 62, 62, 128)	73856
conv2d_5 (Conv2D)	(None, 62, 62, 128)	147584
batch_normalization_2 (Batch	(None, 62, 62, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 31, 31, 128)	0
dropout_2 (Dropout)	(None, 31, 31, 128)	0
flatten (Flatten)	(None, 123008)	0
dropout_3 (Dropout)	(None, 123008)	0
dense (Dense)	(None, 3)	369027

Total params: 656,931
Trainable params: 656,483
Non-trainable params: 448

Figure 2: Potato Model Parameters



(a)



(b)

Figure 3: Potato (a) model loss graph (b) model accuracy graph

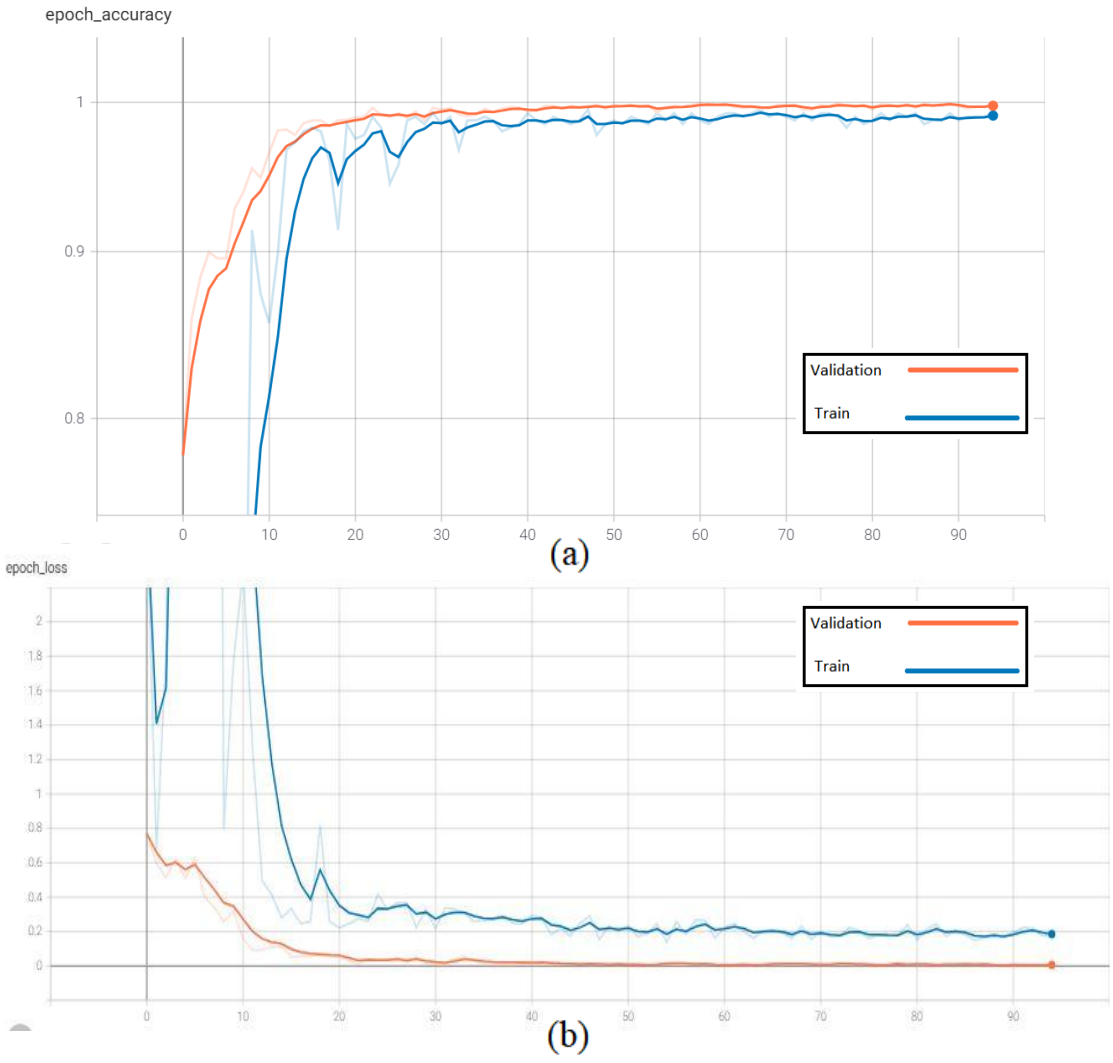
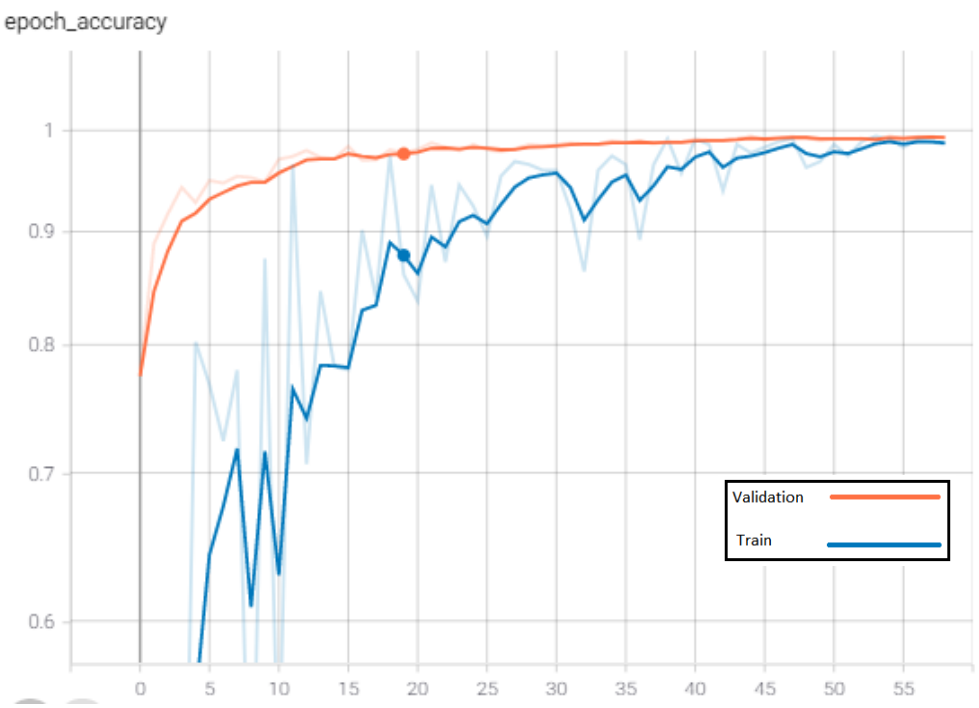
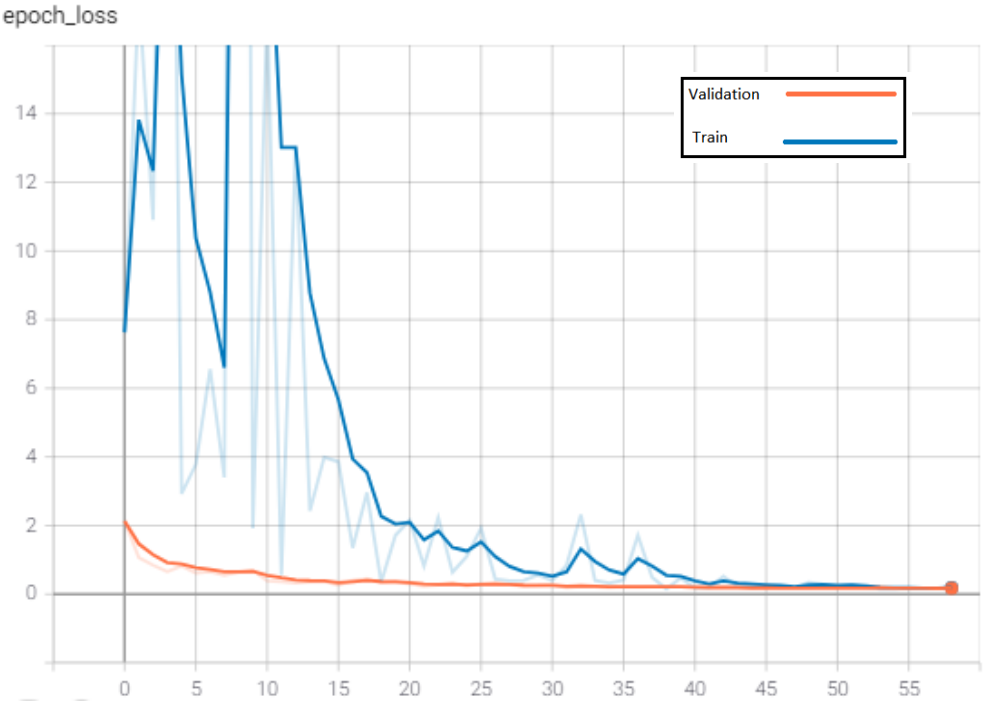


Figure 4: Bell pepper model (a) model loss graph (b) model accuracy graph


epoch_accuracy
1
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30
35
40
45
50
55
Validation
Train

(a)


epoch_loss
14
12
10
8
6
4
2
0
0
5
10
15
20
25
30
35
40
45
50
55
Validation
Train

(b)

Figure 5: Tomato model (a) model loss graph (b) model accuracy graph

Table 1.A) Summary of TALC metrics for the potato, bell pepper and tomato models with Cross-Entropy Loss

Model	**Dataset**	**Accuracy**	**Recall**	**Precision**	**F1 Score**
Bell Pepper	PlantVillage	0.98	0.97	0.97	0.97
Potato	PlantVillage	0.98	0.90	0.90	0.90
Tomato	PlantVillage	0.99	0.95	0.94	0.94
Bell Pepper	Generalized	0.78	0.70	0.64	0.62
Potato	Generalized	0.50	0.49	0.67	0.47
Tomato	Generalized	0.30	0.21	0.13	0.17

Table 1.B) Summary of TALC metrics for the potato, bell pepper and tomato models with Macro Double Soft F1 Loss

Model	**Dataset**	**Accuracy**	**Recall**	**Precision**	**F1 Score**
Bell Pepper	PlantVillage	0.99	0.98	0.98	0.97
Potato	PlantVillage	0.18	0.20	0.17	0.13
Tomato	PlantVillage	0.97	0.92	0.92	0.92
Bell Pepper	Generalized	0.73	0.73	0.58	0.60
Potato	Generalized	0.19	0.19	0.44	0.18
Tomato	Generalized	0.35	0.25	0.16	0.19

Team contributions: Final Report and Presentation

Yuqing (Lisa) Zhao

- Did meeting minutes for the group
- Did general research on DataImageGenerator for the group
- Discussion on Potato model, preliminary results and problem section.
- Introduction, and general editing of report

Kasha Talaga

- Revised the dataset and approach sections and completed the tomato model/results sections.
- In the presentation, I worked on the dataset, approach and tomato model/results slides.
- Contributed to the final tomato model and SSL algorithm.
- Added suggestions and final edits to the document and presentation.

Reza Jafari

- Work on bell-pepper model and try and run the model for different kind of parameters, and discuss the result with the group mate
- Actively participate in group discussion
- Write the motivation part of the research and the reason why this project is important

Evan Boerchers

- Wrote preprocessing sections for presentation and report
- Wrote about generalization of the model
- Worked on bell pepper model
- Improved generalization of bell pepper model

Hunter Kimmett

- Wrote metrics and TALC section of report and presentation
- Ran all models on TALC
- Worked on potato model

Name	Consensus Score
Yuqing (Lisa) Zhao	3
Kasha Talaga	3
Reza Jafari	3

Evan Boerchers	3
Hunter Kimmett	3