# Leaf Wilting Detection in Soybean Crops

Shubham Miglani Tianji Gao Mohammad Sareeb Hakak

## I. METHODOLOGY

### A. Data Acquisition

The data provided for this project is in the form of images with labels defined separately. The labels were sorted to match the images. Google Colab was the choice of the working environment due to the GPU support and high RAM availability. Keras built on TensorFlow was the preferred framework of implementation. The image loader in Keras was used to load the images. Since the image size is large, the RAM got exhausted continuously. To tackle this, we used the Image data generator feature in Keras which loads the images in batches. This makes sure that the RAM is managed properly.

However, the Keras Image data generator expects labeled images to be available in a certain folder hierarchy, the folders being subdivided into different classes. So, the images were divided into 5 classes and split into training (702 images) and validation (194 images) sets. Also, the images were resized to a square image of 124x124.

### B. Scheduled Plan/ Approach

The proposed plan of work was to learn from the experience of Project C1 and learn from it. For C1, we had developed a CNN. The CNN developed gave us an accuracy of 70% for the testing set and it was clear, we needed something more since our data is small. We decided to work with transfer learning. With this, gained knowledge from one problem is applied to a different but related problem.

In transfer learning, very deep networks are already trained with extensive hardware, and from these pre-trained weights as initializations or a fixed feature extractor help in solving most of the problems in hand.

For the project, we decided on using the VGG-19 pretrained CNN. The network has 19 deep layers and can classify 1000 object categories. This makes it an obvious choice for the crop detection project as no pretrained network has classified plants, but VGG-19 has learned rich feature representations for a wide range of images. [1]

With the pretrained model, we have the option to retrain the layers of the network on our data after the VGG-19 is implemented on the data set. The layers for retraining can be chosen from none to all. We had to set some layers as non-trainable which were set to non-trainable i.e. frozen. On top of VGG-19, we implemented two dense layers and the output layer. After the CNN is built, optimizing the hyperparameters and tuning the trainable layers of VGG-19 gives the best-fine-tuned output.

## II. MODEL TRAINING AND HYPER-PARAMETER SELECTION

The initial structure of our neural network was based on the VGG-19 model and the dense layers built on top. The VGG-19 model was fine-tuned by retrains some layers and checking the performance. We set some layers as false trainable, meaning all except them would be retrained. These are the frozen layers. This was taken as the first hyperparameter.

For more hyperparameter selection, the variables which we chose to optimize are the dropout value, the number of neurons in the dense layer, the learning rate, and the L2 regularization value. First, the number of neurons was varied, the best value was selected and updated for the next hyperparameter and so on. The default hyperparameter values are shown in Table1.

Table 1
Default Hyperparameter Values

| Hyperparameter | Default value |
|---|---|
| Neuros in Dense Layer | 512 |
| No. of Frozen Trainable Layers | 15 |
| Learning Rate | 0.0001 |
| Dropout Probability | 0.4 |
| Activation Function | ReLU |
| L2 value | 0.001 |

The range of hyperparameters checked and the best values are shown in Table

Table *2*
Range of Hyperparameter Values

| Hyperparameter | Range Checked | Best Value |
|---|---|---|
| No. of Frozen Trainable Layers | 5,7,9,11,13,15 | **10** |
| Neuros in Dense Layer | 64,128,256,512 | **512** |
| Learning Rate | 7.64e-06, 4.38e-05, 5.38e-05, 7.23e-05, 7.799e-05, 9.77e-05 | **7.99e-05** |
| L2 Regularization value | 0.001,0.01,0.1,0.4 | **0.4** |
| Dropout Probability | 0.30,0.40,0.50,0.60 | **0.4** |

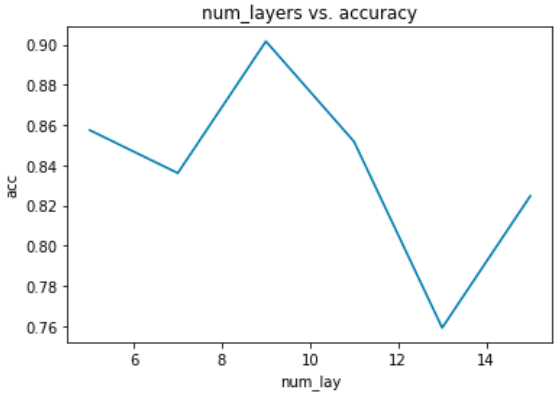
It was also noticed that the validation accuracy was dropping after some epochs and there was not a constant reduction, so the maximum validation accuracy was recorded as well to be used along with early stopping.

*A. Number of Frozen Trainable Layers*

The first thing we started with tuning in the network was choosing how many layers to retrain from the original deep VGG-19 network. The non-trainable layers were chosen as frozen and we tried to tune this as a hyper-parameter.

Table 3
Validation Accuracy for change in neurons in the dense layer

| Number of Frozen layers | Max Validation accuracy |
|---|---|
| 5 | 0.86082 |
| 7 | 0.84536 |
| 9 | 0.88032 |
| 11 | 0.86082 |
| 13 | 0.85567 |
| 15 | 0.83505 |

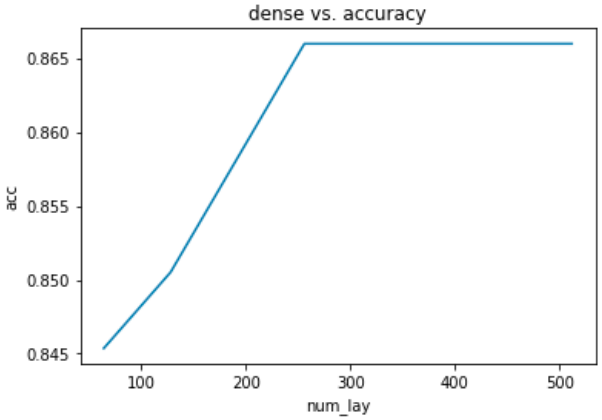



*B. Number of neurons in the hidden layer*

Firstly, the number of neurons in the dense layer was varied. The accuracy remains the same after 256 neurons. The best value chosen was for 512 neurons.

Table 4
Validation Accuracy for change in neurons in the dense layer

| Number of neurons | Max Validation accuracy |
|---|---|
| 64 | 0.8453 |
| 128 | 0.85052 |
| 256 | 0.86598 |
| **512** | **0.86598** |

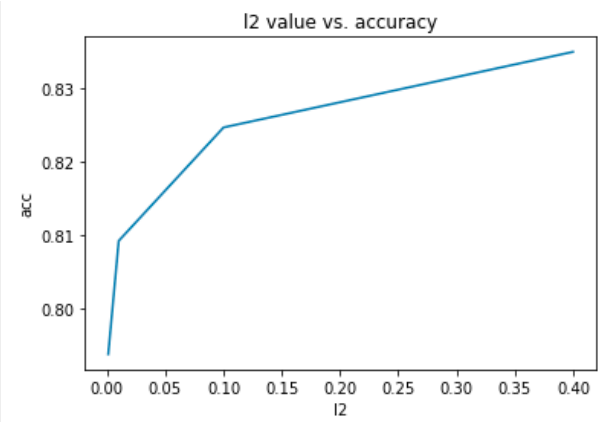



*C. L2 Regularization Valu2*

For the two dense layers, L2 regularization was applied along with dropout since the model was overfitting a lot. The best value came out to be high at 0.4.

Table 5
Validation Accuracy for change in Feature map Size

| L2 value for dense layers | Max Validation accuracy |
|---|---|
| 0.001 | 0.7938 |
| 0.01 | 0.80928 |
| 0.1 | 0.82474 |
| **0.4** | **0.83505** |

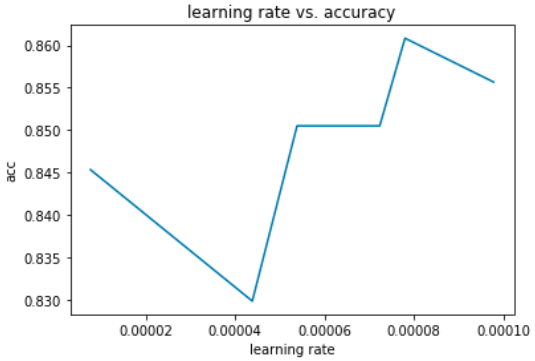



*D. Learning rate for Adam Optimizer*

Thirdly, the learning rate was varied. The curve for validation accuracy is as shown:

Table 6
Validation Accuracy for change in Learning Rate

| Learning Rate | Max Validation accuracy |
|---|---|
| 7.64e-06 | 0.8453 |
| 4.38e-05 | 0.82990 |
| 5.38e-05 | 0.85052 |
| 7.23e-05 | 0.8505 |
| **7.799e-05** | **0.86082** |
| 9.77e-05 | 0.85567 |



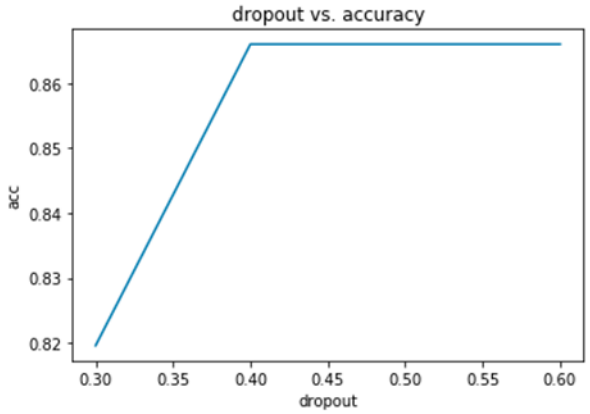

Although the max validation accuracy for learning rate is fluctuating, this is based on a random generator and we get the

best accuracy for 7.799e-05 which is chosen for the final model.

### *E. Dropout Probability*

Table 7
Validation Accuracy for change in Dropout

| Dropout | Max Validation accuracy |
|---|---|
| 0.30 | 0.81959 |
| 0.40 | 0.86598 |
| 0.50 | 0.86598 |
| 0.60 | 086598 |




The accuracy remains the same for values after 0.40 and is chosen as the final value for the network. Max norm was also decided to be used based on [4] to reduce the variance between validation and training accuracy.

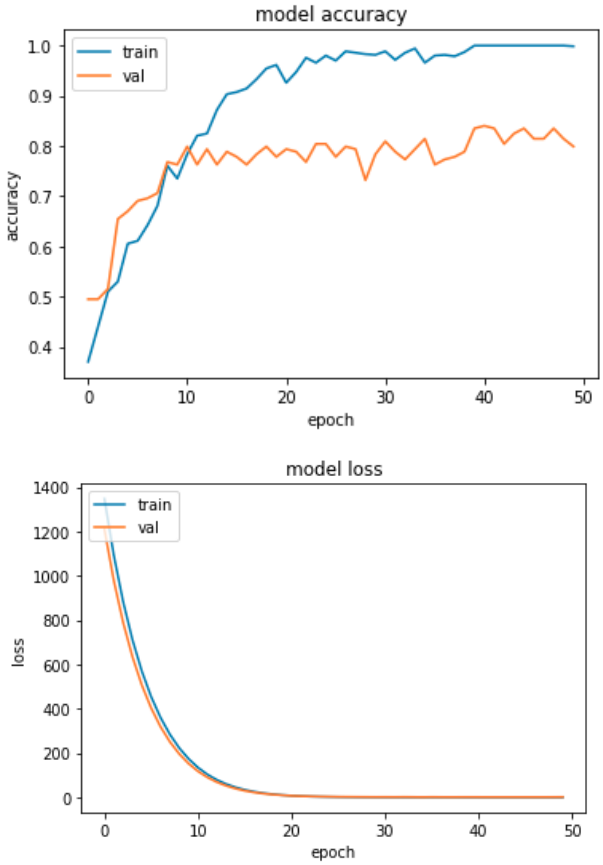
The final structure of our network is:

Table 8
Final Network Structure

| Layer type | Output Shape | Parameters |
|---|---|---|
| Conv2D | (None,128,128,64) | 1792 |
| Conv2D | (None,128,128,64) | 36928 |
| Max Pooling 2D | (None,64,64,64) | 0 |
| Conv2D | (None,64,64,128) | 73856 |
| Conv2D | (None,64,64,128) | 147584 |
| Max Pooling 2D | (None,32,32,128) | 0 |
| Conv2D | (None,32,32,256) | 295168 |
| Conv2D | (None,32,32,256) | 590080 |
| Conv2D | (None,32,32,256) | 590080 |
| Conv2D | (None,32,32,256) | 590080 |
| Max Pooling 2D | (None,16,16,256) | 0 |
| Conv2D | (None,16,16,512) | 1180160 |
| Conv2D | (None,16,16,512) | 2359808 |
| Conv2D | (None,16,16,512) | 2359808 |
| Conv2D | (None,16,16,512) | 2359808 |
| Max Pooling 2D | (None,16,16,512) | 0 |
| Conv2D | (None,8,8,512) | 2359808 |
| Conv2D | (None,8,8,512) | 2359808 |
| Conv2D | (None,8,8,512) | 2359808 |
| Conv2D | (None,8,8,512) | 2359808 |
| Max Pooling 2D | (None,4,4,512) | 0 |
| Dense | (None,512) | 4194816 |
| Dense | (None,512) | 262656 |
| Output | (None,5) | 2565 |

## III. EVALUATION

The learning curves for the final model are as shown:




The network chosen had a validation loss of **1.0791** and validation accuracy of **0.835051**
The error metrics are as follows:

Table 9
Error Metrics for Transfer Learning

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.86 | 0.93 | 0.89 | 96 |
| 1 | 0.79 | 0.58 | 0.67 | 33 |
| 2 | 0.67 | 0.83 | 0.74 | 24 |
| 3 | 0.88 | 0.84 | 0.86 | 25 |
| 4 | 1.00 | 0.81 | 0.90 | 16 |

## IV. SEMI-SUPERVISED LEARNING

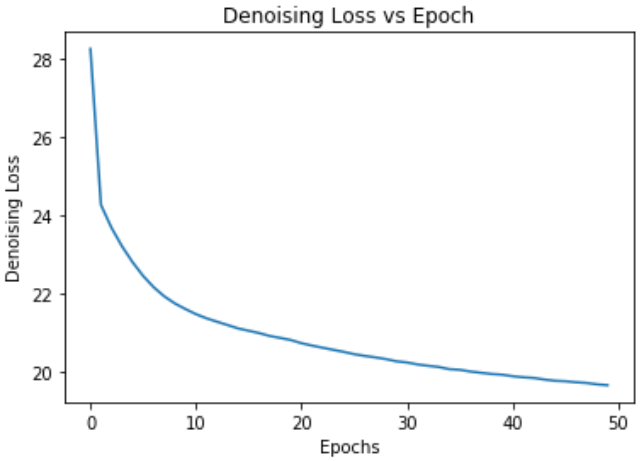
The unlabeled images that were given were manually filtered for all the bad data. Further, training data and test images were also added to the same folder. After the filtering process, a total of 15,444 images were used for the unlabeled ones. The distribution of validation and training set was the same as before (702 and 194). The data was converted to npy format and then could be used much more efficiently as the images won't have to be loaded again and again. Also reading data from npy format is much faster.

For semi-supervised learning, we took reference from [5] where supervised learning is combined with unsupervised learning. The model is trained to simultaneously minimize the

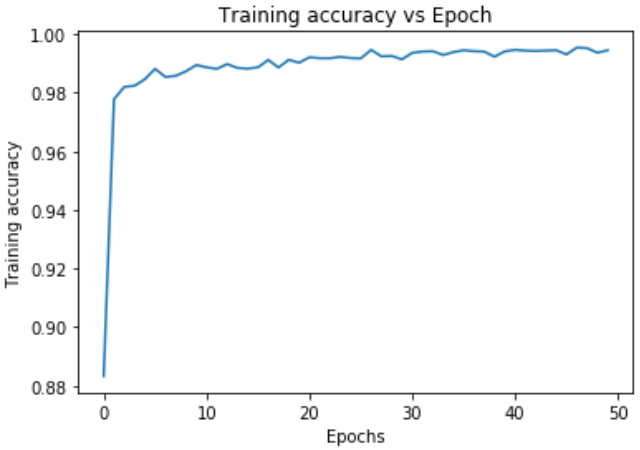
losses of both supervised and unsupervised parts by building on the Ladder network.

The model structure is an autoencoder with skip connections from the encoder to the decoder and the learning task is similar to denoising autoencoders but applied to every layer and not just the inputs.

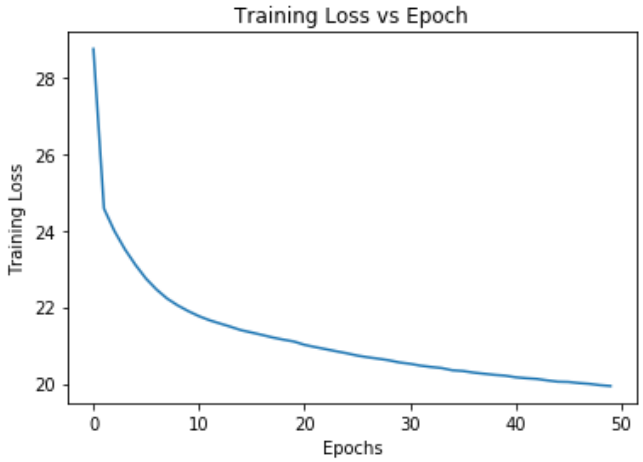
The model structure in summary was Dense layers of [128*128*3(input image size), 1000, 500, 250, 250, 250, number of classes(5)]. A standard deviation of 0.3 was used for the noise along with denoising costs of [1000, 10, 0.1, 0.1, 0.1, 0.1, 0.1] corresponding to the dense layers defined earlier. Due to a large number of images and slow training, hyperparameter tuning, in this case, was not done but the default parameters from [5] were used to train the model. The model was trained for 50 epochs and the results were as follows:
The denoising loss for the unlabeled images was as follows:




The training accuracy for the labeled images is as follows:




The training loss for the labeled images is as follows:




The final validation accuracy achieved for this case was 0.835. The error metrics are as follows:

Table 10
Error Metrics for Semi-Supervised Learning

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.87 | 0.92 | 0.89 | 101 |
| 1 | 0.67 | 0.65 | 0.66 | 31 |
| 2 | 0.90 | 0.72 | 0.80 | 25 |
| 3 | 0.85 | 0.92 | 0.88 | 24 |
| 4 | 0.82 | 0.69 | 0.75 | 13 |

The testing accuracy we achieved using this was 75.1% higher than 71% achieved in the C1 phase.
Also, to note that the model doesn't overfit because of the noise being added so it can be trained for many more epochs. There was also a suggestion in the paper to use CNN's along with autoencoders for improving performance which we were not able to implement but according to the paper will lead to an improvement in the performance of the network.

## REFERENCES


[1] https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8

[2] B. Zhong, Q. Ge, B. Kanakiya, R. Mitra, T. Marchitto, E. Lobaton, "A Comparative Study of Image Classification Algorithms for Foraminifera Identification," IEEE Symp. Series on Computational Intelligence (SSCI), 2017.

[3] Serawork Wallelign, Mihai Polceanu, Cedric Buche, "Soybean Plant Disease Identification Using Convolutional Neural Netowrk" Artificial Intelligence Research Society Conference (FLAIRS-31).

[4] Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15, no. 1 (2014): 1929-1958.

[5] arXiv:1507.02672v2 [cs.NE]

[6] https://github.com/divamgupta/ladder_network_keras