Leaf Wilting Detection in Soybean

Archit Kwatra
200316390
akwatra@ncsu.edu

Piyush Tiwari
200314347
ptiwari@ncsu.edu

Jay Jagtap
200311438
jjjagtap@ncsu.edu

I. Methodology

We have used a CNN based model for leaf wilting detection in soybean plants which has been a widespread problem due to weather stress and climate changes. We have a used a training data of 1025 soybean plant's images with 5 classes of labels. To predict the image labels, we have used convolutional neural networks, with multiple layers. To train our neural network, the image is fed to the input layer with its original shape of 640 x 480 x 3, where 640 and 480 is the size of the image in pixels and 3 represents the RGB property of the image. We have used tensorflow-keras for building our model.

II. Model Training and Hyperparameter Selection

A. Load data and Preprocess

we split our data into 70% training, 20% validation and 10% testing for training, hyper-parameter tuning and testing our model.

B. Hyper-Parameter Selection

We have used keras' imageDataGenerator to preprocess and feed our images to the input layer. To carry out the preprocessing, we initially set the shear_range and zoom_range in the imageDatagenerator function to value of 0.1 and 0.3 respectively, where shear_range is the shear angle in counter-clockwise direction in degrees and zoom_range is the range for random zoom of an image. However, we got better results by setting the shear_range and zoom_range to 0.2, Also, we have set horizontal_flip to true, which randomly flips an image horizontally to get better results.

We have used imageDataGenerator to preprocess the images because the data set is very less and we want to train our model in batches, taking this batches and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.). Replacing the original batch with the new, randomly transformed batch. Training the CNN on this randomly transformed batch (i.e., the original data itself is not used for training) so that we can get generalized result.

To train our model we have used keras fitGenerator function to fit our model on the training data. To get the maximum accuracy, we initially set the number of epochs to 10 and plotted a graph by gradually increasing the epochs' values and the best accuracy was obtained at 32 epochs. Once, we were able to obtain the number of epochs, we varied the batch size for stochastic gradient calculation which ranged from 16 to 64. The best results were obtained at a value of 32 for the batch size. We know that these settings give good results with CNN and have not experimented much with them. We made sure that we have run our model for enough epochs to make our coverage most accurate with the given data.

We have used 3 convolutional layers with 'relu' activation and done maxpooling, which has a poolsize of (3x3) for the first two layers and (2x2) for the last two layers, in all the layers with a stride of 2. We experimented with multiple window sizes and chosen the one which gave the best results.

To speed up the model we have used Adam optimizer with learning rate learning_rate 0.001, keeping the beta_1 and beta_2 value 0.9 and 0.999 respectively.

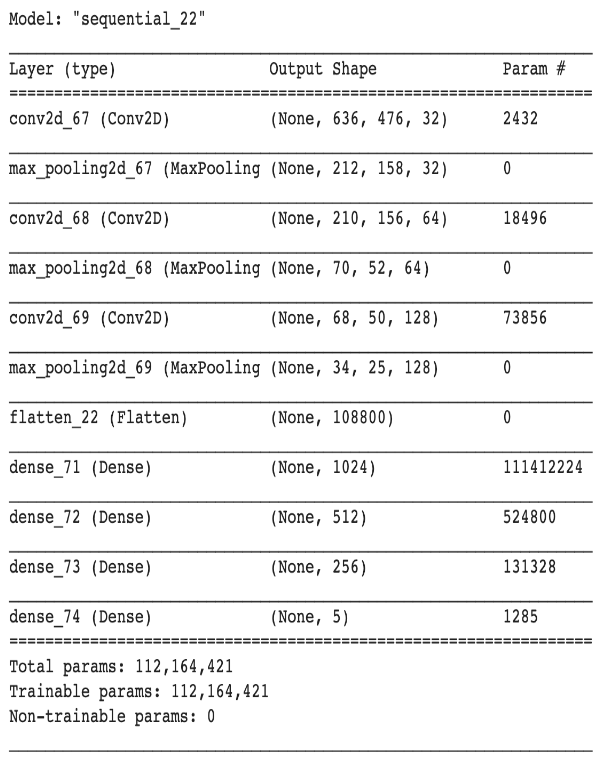
Similarly, we also tried different number of features at convolutional layers. Expressiveness of model gets better if we increase the number of features at each layer. But if we keep increasing the number of features, there would come a point where different features would be redundant as there is nothing to learn and each feature has very little information. Again we tried with the numbers given in fig[1].we take a design choice of [640, 480] features in [conv1]. We have chosen this number because after [640, 480], we observe that accuracy is not increasing significantly.

we have used 4 dense layer in our model with dimension of [1024, 512, 256, 5]. The activation function in the first three layer is Relu and in the last layer we have used the softmax function.

We show the validation accuracy with different layers and epochs in the following table.

Experiments		
Layers	Epoch	Accuracy
2	25	81.85
3	25	84.64
3	50	85.98
2	50	83.76
1	25	79.21
1	50	81.38

The Final Model summary is defined in Fig1.



```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_67 (Conv2D)           (None, 636, 476, 32)      2432
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 212, 158, 32)      0
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 210, 156, 64)      18496
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 70, 52, 64)        0
_________________________________________________________________
conv2d_69 (Conv2D)           (None, 68, 50, 128)       73856
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 34, 25, 128)       0
_________________________________________________________________
flatten_22 (Flatten)         (None, 108800)            0
_________________________________________________________________
dense_71 (Dense)             (None, 1024)              111412224
_________________________________________________________________
dense_72 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_73 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_74 (Dense)             (None, 5)                 1285
=================================================================
Total params: 112,164,421
Trainable params: 112,164,421
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. Model Summary

III. Evaluation

Each image is fed as an input of 640*480*3 (color image) and all the pixel values are normalized in the range of 0 to 1 from 0 to 255 as convolutional neural networks are known to work better with normalized values. We divide the 1025 images in training set, validation set and test set to tune our hyper-parameters. We use stratified data sampling to ensure proportional representation of each class in all 3 data sets. We used the accuracy on validation data set to tune our hyper-parameters. Finally, after hyper-parameter tuning, we achieved an accuracy of 85.9 percent on the test data set. The accuracy graph between training and validation data is shown in fig[2] and validation loss is shown in fig[3].

The final model with chosen hyper parameters is given in the table

Hyper-Paramters For Convolutional Layer			
	Layer1	Layer2	Layer3
Filter size	32	64	128
Feature	(636, 476, 32)	(210, 156, 64)	(68, 50, 128)
Activation	Relu	Relu	Relu

Hyper-Paramters For Dense Layer				
	Layer1	Layer2	Layer3	Layer4
units	1024	512	256	5
Activation	Relu	Relu	Relu	Softmax
kernel regularizer	0.01	0.01	0.02	0.02

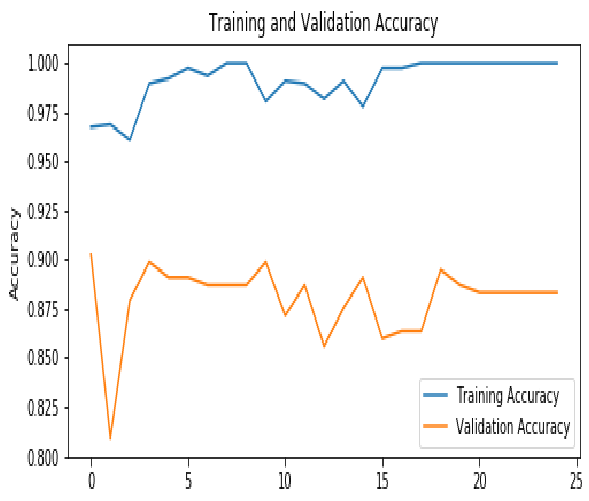


Fig. 2. Training and validation Accuracy

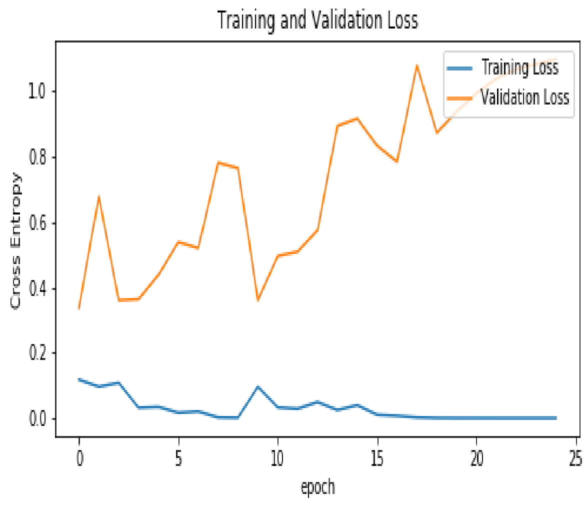


Fig. 3. Training and validation loss

Given below (Fig. 4) is the plot showcasing the inference (prediction) and ground-truth plot.

Here, we can see that the ground-truth values are represented in bright red and the prediction is represented blue in color. The darker red colored points showcasing overlap of the red and blue colors denote correct prediction. In the plot given, we can see that in the window that is being considered, only few false predictions are noted since there are few occurrences where the prediction and ground-truth points do not overlap.

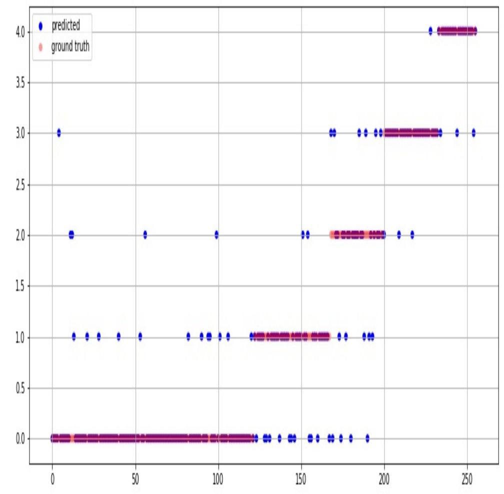


Fig. 4. Inference and Ground Truth

References

[1] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional neural networks. arXiv preprint arXiv:1311.2901 (2013)