Leaf Wilting Detection in Soybean

Abhirav Kariya (akariya)
North Carolina State University
akariya@ncsu.edu

Nisarg Chokshi (nmchoks2)
North Carolina State University
nmchoks2@ncsu.edu

Abhinav Kashyap (akashya3)
North Carolina State University
akashya3@ncsu.edu

I. Methodology

For the classification problem of Leaf Wilting Detection in Soybean, we implemented a network architecture as shown in fig. 1, consisting of two parallel and independent networks for feature extraction, namely Xception [1] and MobileNetV2 [2]. Both are followed by independent fully-connected networks. Finally, there is a combination layer where the two modules meet. We train all the layers of both the architectures while using the pre-trained weights only as initialization. We train the model after converting the images into the HSV color space and use an input shape of 299*299*3 for both architectures.

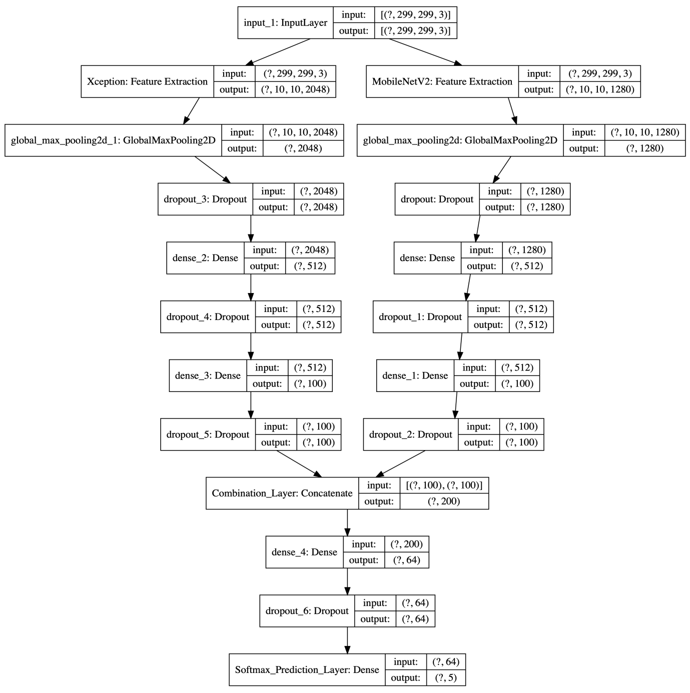

input_1: InputLayer | input: [(?, 299, 299, 3)] | output: [(?, 299, 299, 3)]
Xception: Feature Extraction | input: (?, 299, 299, 3) | output: (?, 10, 10, 2048)
MobileNetV2: Feature Extraction | input: (?, 299, 299, 3) | output: (?, 10, 10, 1280)
global_max_pooling2d_1: GlobalMaxPooling2D | input: (?, 10, 10, 2048) | output: (?, 2048)
global_max_pooling2d: GlobalMaxPooling2D | input: (?, 10, 10, 1280) | output: (?, 1280)
dropout_3: Dropout | input: (?, 2048) | output: (?, 2048)
dropout: Dropout | input: (?, 1280) | output: (?, 1280)
dense_2: Dense | input: (?, 2048) | output: (?, 512)
dense: Dense | input: (?, 1280) | output: (?, 512)
dropout_4: Dropout | input: (?, 512) | output: (?, 512)
dropout_1: Dropout | input: (?, 512) | output: (?, 512)
dense_3: Dense | input: (?, 512) | output: (?, 100)
dense_1: Dense | input: (?, 512) | output: (?, 100)
dropout_5: Dropout | input: (?, 100) | output: (?, 100)
dropout_2: Dropout | input: (?, 100) | output: (?, 100)
Combination_Layer: Concatenate | input: [(?, 100), (?, 100)] | output: (?, 200)
dense_4: Dense | input: (?, 200) | output: (?, 64)
dropout_6: Dropout | input: (?, 64) | output: (?, 64)
Softmax_Prediction_Layer: Dense | input: (?, 64) | output: (?, 5)

Fig. 1: Network Architecture

We do not change the activations in the CNN architectures that we have used. In the final fully-connected combination layers we use LeakyRELU [10] as the activation function. The choice of optimizer is Adam [12]. The choice for loss function is categorical cross-entropy.

A. Xception

Xception architecture [7] as shown in fig. 2 is inspired by Google's Inception Model [5] and its architecture is a linear stack of depth-wise separable convolution layers with residual connections [3]. Depth-wise separable convolution involves looking at the channel and spatial correlations in successive steps instead of looking at them simultaneously. This reduces the parameter count and increases efficiency. The architecture consists of 36 convolutional layers for feature extraction structured into 14 modules, all of them having linear residual connections except for the first and last modules.

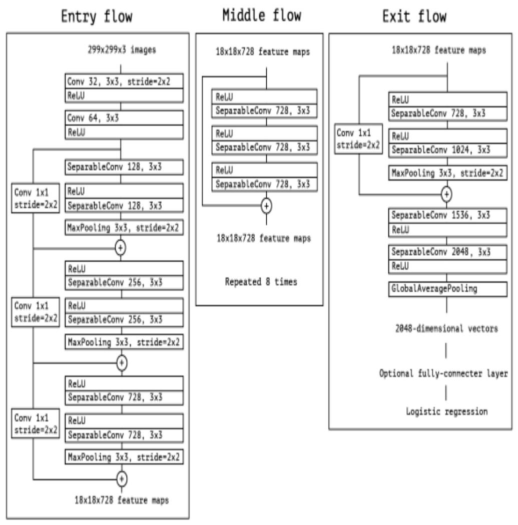


Fig. 2: Xception Architecture

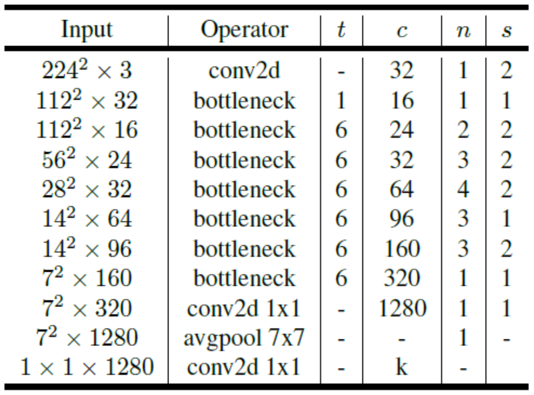
B. MobileNetV2

The MobileNetV2 [2] architecture as shown in fig. 3 is based on a bottleneck depth-separable convolution with residual connections. The architecture of MobileNetV2 contains the initial convolutional layer with 32 filters, followed by 19 residual bottleneck layers. The top layers are not included while importing this architecture. We trained this model on images of dimensions 299*299*3 instead of the default 224*224*3.

Note: For both these architectures, we use the ImageNet [11] pretrained weights made available by the Keras library [14] just as initial weights and train all the layers. We add high L1 regularization to all the layers in order to achieve sparsity, consequently trying to avoid over-fitting.

C. Data Set

The given data set consists of images of soybean plants and the labels for the level of wilting. The images in the



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Fig. 3: MobileNetV2 Architecture [8]

data set are taken at different times in the day and hence vary in intensity. It is important to make sure that the model does not learn such undue characteristics and attribute them to classes. Furthermore, The training set is a bit skewed and has fewer examples of class 2 and class 3. We use various data augmentation strategies to combat these problems. The said strategies are explained briefly in the next section.

II. MODEL TRAINING AND SELECTION

A. Model Training

Before we apply any augmentation strategies, we split the data set into a training and validation set. The said validation set consists of a perfectly balanced class distribution. The train to validation ratio is set to 4:1.

In order to offset the class imbalance, we have used some data augmentation strategies. These strategies take into consideration that finally want to use 299*299*3 dimensional images. We first resize images to 360*480*3 dimension and apply the following strategies in order.

- Augmentation by cropping: We generate 5 cropped patches of 299*299*3 dimension from the resized image by extracting 4 cropped patches from 4 corners and 1 cropped patch from the center. Then we randomly sample a more balanced data set (not completely balanced).
- Augmentation by transformation: From the sampled data set, we select 10 % of the images, 5 % are blurred and 5 % are sharpened and these modified images are added to the data set.
- Augmentation using ImageDataGenerator [9]: While training we use the ImageDataGenerator that randomly modifies images for every batch according to the given parameters. We use techniques like flipping, rotation, zoom et cetera.

Note: All these procedures are executed during every run independently and the augmented set is, thus, not static.

As we are using ImageDataGenerator [9], we choose to sample-wise center and sample-wise normalize every input image during training.

We use the EarlyStopping callback provided by the Keras library and set patience of 3 epochs. Furthermore, if EarlyStopping is in fact triggered, we restore the weights from the best epoch (taking validation loss into account).

We use class weights [13] defined as:

$$Weight\,Class_i = \frac{Total\,Samples}{2 * Total\,Samples\,of\,Class_i}$$

We initialize prediction layer bias [13] as:

$$P_i = \frac{Total\,Samples\,of\,Class_i}{Total\,Samples - Total\,Samples\,of\,Class_i}$$

$$B_i = \log P_i$$

B. Model Selection

We unfreeze all the layers (or rather, do not freeze any layers) of both the architectures (Xception [1] and MobileNetV2 [2]) while training the model. Since our model is very deep, our parameter count is very high. To combat over-fitting we extensively regularize our model. We use different strategies in the architectures that we use for feature extraction and in the fully-connected layers that we attach after them. In both cases we penalize the activity of neurons rather than penalizing bias and kernels separately.

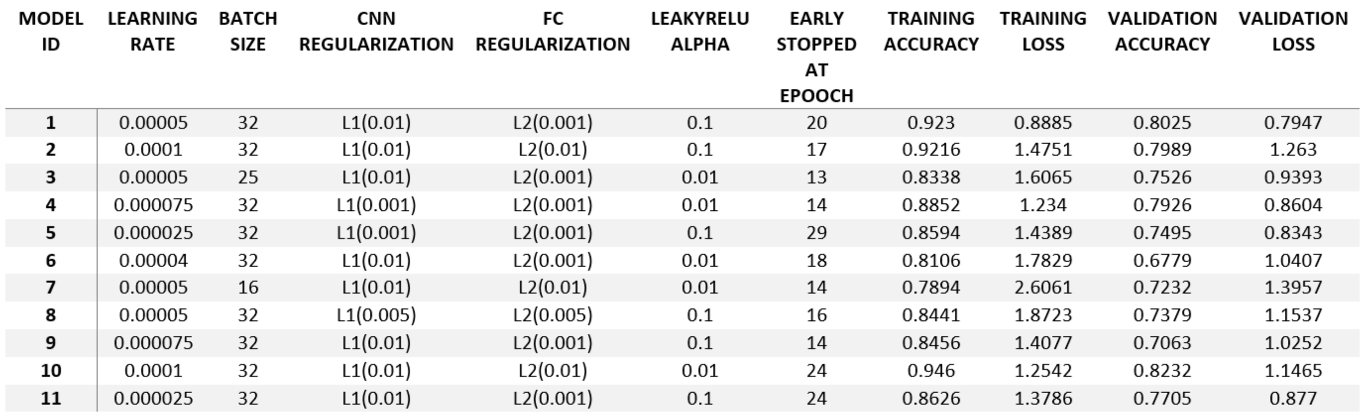
Some of the models and their performance that we observed in order to tune hyper-parameters are presented in table I

As we use randomly selected cropped patches for training and evaluate the models on accuracy at the patch level, there is a need for an ensemble model to finally evaluate the validation accuracy (and to predict on the test set). As we take five patches from every image and train the model with these patches, we then have five candidate models for an ensemble model (one each corresponding to the crop location in the original image). The ensemble strategy that works best on the validation data will be used on test data. The strategies that we tried are as follows (we have 5 classifiers for a possible ensemble):

- Majority voting: Take a majority vote amongst the classifiers.
- Majority voting (Top 3 by confidence): We create 3 latent classifiers by sorting the predicted probabilities per class and picking the highest 3 probabilities. Then, we take a majority vote.
- Max confidence: As we have 25 probabilities (5 per class per classifier) we look at the most confident prediction out of these 25 probabilities and choose the corresponding class as the prediction.
- Total confidence per class: We add the probabilities per class from all the classifiers and choose the class with the maximum volume as the classification.
- Total confidence per class (Top 3 by confidence): Instead of adding all the probabilities from all 5 classifiers, we just add the top 3 probabilities to compute the volume for that class. Then we choose the class with the maximum volume as the prediction.

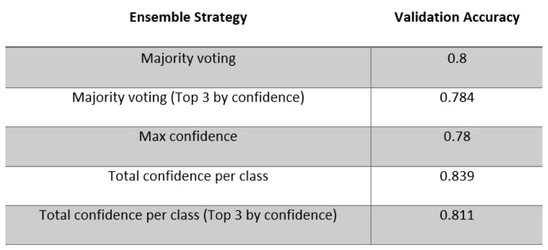
The validation accuracies for the above listed ensemble strategies is listed in the table II.

TABLE I: Model Selection



MODEL ID	LEARNING RATE	BATCH SIZE	CNN REGULARIZATION	FC REGULARIZATION	LEAKYRELU ALPHA	EARLY STOPPED AT EPOOCH	TRAINING ACCURACY	TRAINING LOSS	VALIDATION ACCURACY	VALIDATION LOSS
1	0.00005	32	L1(0.01)	L2(0.001)	0.1	20	0.923	0.8885	0.8025	0.7947
2	0.0001	32	L1(0.01)	L2(0.01)	0.1	17	0.9216	1.4751	0.7989	1.263
3	0.00005	25	L1(0.01)	L2(0.001)	0.01	13	0.8338	1.6065	0.7526	0.9393
4	0.000075	32	L1(0.001)	L2(0.001)	0.01	14	0.8852	1.234	0.7926	0.8604
5	0.000025	32	L1(0.001)	L2(0.001)	0.1	29	0.8594	1.4389	0.7495	0.8343
6	0.00004	32	L1(0.01)	L2(0.001)	0.01	18	0.8106	1.7829	0.6779	1.0407
7	0.00005	16	L1(0.01)	L2(0.01)	0.01	14	0.7894	2.6061	0.7232	1.3957
8	0.00005	32	L1(0.005)	L2(0.005)	0.1	16	0.8441	1.8723	0.7379	1.1537
9	0.000075	32	L1(0.01)	L2(0.001)	0.1	14	0.8456	1.4077	0.7063	1.0252
10	0.0001	32	L1(0.01)	L2(0.01)	0.01	24	0.946	1.2542	0.8232	1.1465
11	0.000025	32	L1(0.01)	L2(0.001)	0.1	24	0.8626	1.3786	0.7705	0.877

TABLE II: Ensemble Selection



Ensemble Strategy	**Validation Accuracy**
Majority voting	0.8
Majority voting (Top 3 by confidence)	0.784
Max confidence	0.78
Total confidence per class	0.839
Total confidence per class (Top 3 by confidence)	0.811

Note: The accuracies presented in table II for ensemble strategies are computed for the final model (architecture and hyper-parameters) that we use.

III. Evaluation

We selected model 1 (from table I) as our final model.

A. *Model Specification*

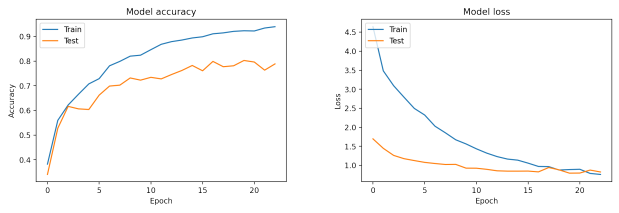
Our Criteria for a better model is validation loss. As tabulated in table I, model 10 has higher accuracy, however, it also has a higher validation loss. As we are using categorical cross-entropy, a higher loss would mean that the model is not confident about it's predictions. Keeping this in mind, model 1, which has a lower loss and lower accuracy than model 10 is able to make, comparatively, make more confident decisions on the validation set.

Final model: The architecture for our model can be seen in fig. 1 We train the final model with hyper-parameters as described for model 1 in table I and use 'Total confidence per class' ensemble strategy as described in the previous section.

B. *Model Evaluation*

The plots for loss and accuracy can be seen in fig. 4(b) and fig. 4(a) respectively.

Before applying the ensemble strategy, the training accuracy, training loss, validation accuracy and validation loss were 0.923, 0.8885, 0.8025, 0.7947 respectively (at patch level). The validation metrics (after running ensemble strategy) are displayed in table III.



(a) Accuracy (b) Loss

Fig. 4: Model Results

TABLE III: Validation Metrics

Class	Precision	Recall	F1-score	Support
0	0.91	0.84	0.88	51
1	0.61	0.96	0.75	51
2	0.92	0.67	0.77	51
3	0.97	0.73	0.83	51
4	0.94	0.98	0.96	51
Macro Average	0.87	0.84	0.84	
Final Validation Accuracy: 0.839				

Note: The scores documented in table III are after running the ensemble strategy and not on patch level.

C. *Test Results (C2)*

Our model achieved a test accuracy of **61.5%** and a RMSE score of **0.7211**. It was good to see that our model performed well and consistently in both metrics.

IV. EXTRA CREDIT: USING WEATHER DATA

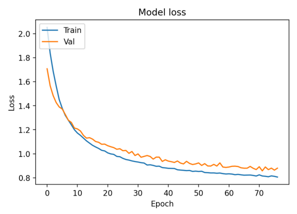
As we have weather data available, it is possible to improve the quality of predictions as wilting has positive correlation with weather conditions. This section will demonstrate a proof of concept to use weather related numerical features along with a CNN. Our approach uses a branched model with a combination layer in the end.

A. *Baseline Models*

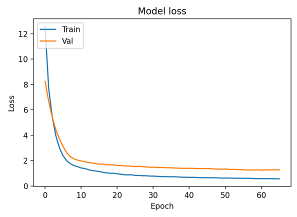
For the weather features model, we decided to make use of a fully-connected neural network. The weather features were normalized by subtracting feature-wise mean and dividing by respective feature-wise standard deviations. A Gaussian Noise layer is added to the network with the assumption that minor changes in the weather parameters will not affect the classification (or the actual effect of the weather on the plants), and hence, will help make the model robust.

For the image data model, we use MobileNet [6] as the base model (without top layer). We attach a fully-connected layer after the feature extraction CNN. To regularize the base model we use high L1 activity regularization.

The loss plots for baseline models can be seen in fig. 5



(a) Weather Model Loss



(b) Image Model Loss

Fig. 5: Baseline Model Loss Plots

B. Model Specification

For the hybrid model, we train the same architectures of the baseline in parallel. The only change that we make is that the last layers have a ReLU activation instead of softmax. Basically, now we have ten features, five from each model. Then we add a combination layer of 5 neurons with a softmax activation. This is the prediction layer. The model architecture can be seen in fig. 6.

The hyper-parameters used for training: **Loss:** Categorical Cross Entropy **Optimizer:** Adam **Learning Rate:** 1e-4 **Batch Size:** 64 **Early Stopping Patience:** 5

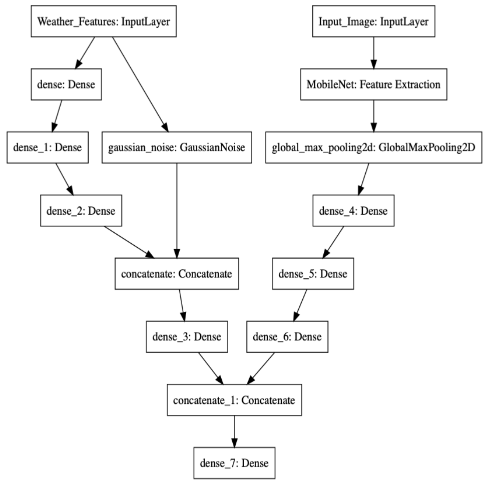
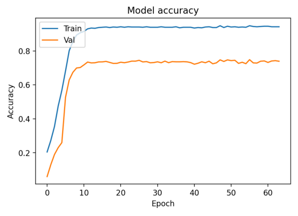


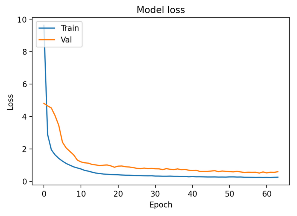
Fig. 6: Hybrid Model Network Architecture

C. Model Evaluation

The final accuracy achieved on the test set was 0.76. The test metrics for the baseline models and the hybrid model can be seen in table IV The plots for the final model can be seen in fig. 7



(a) Accuracy



(b) Loss

Fig. 7: Combined Model Accuracy and Loss plots

TABLE IV: Test Metrics

Model	Class	Precision	Recall	F1-score	Support
Weather Features Model	0	0.52	0.85	0.64	20
	1	0.79	0.75	0.77	20
	2	0.76	0.80	0.78	20
	3	0.88	0.35	0.50	20
	4	0.74	0.70	0.72	20
Macro Average		0.74	0.69	0.68	
Image Data Model	0	0.56	0.90	0.69	20
	1	0.59	0.80	0.68	20
	2	0.45	0.25	0.32	20
	3	0.69	0.45	0.55	20
	4	0.88	0.75	0.81	20
Macro Average		0.64	0.63	0.61	
Hybrid Model	0	0.55	0.85	0.67	20
	1	0.82	0.90	0.86	20
	2	0.88	0.75	0.81	20
	3	0.75	0.45	0.56	20
	4	0.94	0.85	0.89	20
Macro Average		0.79	0.76	0.76	
Final Test Accuracy (Weather Features Model): 0.69					
Final Test Accuracy (Image Data Model): 0.63					
Final Test Accuracy (Hybrid Model): 0.76					

References

[1] Chollet, François. “Xception: Deep Learning with Depthwise Separable Convolutions.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017): 1800-1807.
[2] Sandler, Mark et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018): 4510-4520.
[3] https://stephan-osterburg.gitbook.io/coding/coding/ml-dl/tensorflow/ch3-xception/xception-architectural-design
[4] https://keras.io/applications/xception
[5] https://cloud.google.com/tpu/docs/inception-v3-advanced
[6] https://keras.io/applications/mobilenetv2
[7] https://www.shortscience.org/paper?bibtexKey=journals/corr/1610.02357
[8] https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c
[9] https://keras.io/preprocessing/image/
[10] Xu, Bing et al. “Empirical Evaluation of Rectified Activations in Convolutional Network.” ArXiv abs/1505.00853 (2015): n. pag.
[11] http://www.image-net.org/
[12] Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. 2014. arXiv:1412.6980v9
[13] https://www.curiousily.com/posts/practical-guide-to-handling-imbalanced-datasets/
[14] https://keras.io/