Predicting 6 Vital Plant Traits Using an Ensemble of Models Based on Pre-trained CNNs

Stephen Hwang
School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
s29hwang@uwaterloo.ca

August 12, 2024

Abstract

Given a dataset of size 128×128 images coupled with 163 auxiliary attributes, we propose using neural networks that incorporate various pre-trained convolutional neural networks to predict the real values of 6 target traits. Two separate neural net branches are used to process the image and additional attributes, the former of which incorporates one of Inception-Resnet-V2, Resnet-101, and MobileNet-V2 pre-trained on the ImageNet database. The predictions from each model are then combined to produce the test predictions, which have an R^2 score of 0.19337.

The code for this project can be found here: https://github.com/dreaming-fox/uw-cs-480-spring-2024/tree/main

1 Introduction

We are given a regression task where we need to predict 6 real values according to a series of data points. Each data point in our dataset consists of two primary components. Firstly, we get a size 128×128 image of the plant sampled from the iNaturalist database. Paired with each plant comes with a set of 163 additional real-valued traits (which generally we will denote as x) describing the plant's environmental climate, soil properties, etc. True values for the 6 target traits (which we will generally denote as y) are provided with the training set.

2 Related Works

This project was based off of Schiller et al. 2021's experiment, in which they seek to predict the same 6 traits after formulating a dataset of 512×512 plant photographs, their geolocations, and additional location-based climate information. They utilize convolutional neural nets as their primary model for images paired with a parallel neural network comprised only of linear layers for auxiliary data, combining them using further fully connected layers. They also try an ensemble method, averaging the outputs of three weaker models to produce a more robust prediction.

Dong et al. 2023 also use CNNs for plant-related tasks, and have created a collection of pre-trained models that can be applied to a wide variety of applications. By using transfer learning in conjunction with feature maps, FC layers, and other components, users can train models for classification, object detection, and other tasks. Models can be trained more efficiently and perform with higher accuracy than models trained from scratch.

Importantly, both works mentioned reference pre-trained models as a crucial component of modern computer vision learning. Stored weights for established frameworks (including but not limited to

CNN frameworks like ResNet and Inception) can become a powerful tool if used correctly, allowing models to be trained without much of the computational (and sometimes manual) work.

3 Main Results

In this project, we want to maximize the R^2 coefficient of our test set predictions, defined as for any target trait y:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

3.1 Data Preparation and Pre-Processing

We follow Schiller et al. 2021's general method in most of our dataset preprocessing. Firstly, we remove every datapoint with a target value outside of 3.5 standard deviations from the mean to ensure outliers don't heavily affect our model.

We also perform min-max normalization on each x using the mininum and maximum values from both the training and test set. This was done externally, with the results being included in `data/max-mins.csv`. Formally, let $\tilde{x}$ be the normalized value and min and max the minimum and maximum values of some attribute, then:

$$\tilde{x} = \frac{x - min}{max - min}$$

We do the same thing for each y, resulting in the condition so that both $x, y \in [0, 1]$ for any attribute. As a side note, note that this means we perform the reverse calculation when generating predictions fot the test set. Let $\hat{y}$ the submmited prediction for target y, then:

$$y = \tilde{y}(max - min) + min$$

For the images, we perform a number of transformations at random before converting it to a tensor and normalizing. With a 0.5 chance we flip the image horizontally, adjust the brightness to be within a factor of 0.9-1.1, and randomly zoom in up to a factor of 1.2. To normalize we use the RGB means and standard deviations (i.e. $[0.44636917, 0.45045858, 0.33603736]$ and $[0.21836502, 0.20886066, 0.21879451]$ respectively), as the photos tend to be a bit green (and apparently red) compared to the average photo available on a dataset like Imagenet.

Note that because we use pre-trained CNNs, we scale up the image to size 224×224. While this may lead to a larger input with no additional information, this allows us to fully use the full potential of the pre-trained models.

3.2 Model Structure

Our model (as shown in Figure 1 below) initially trains on the image and attribute values separately. A pre-trained CNN takes in the processed image and passes it to a series of fully connected layers, while the other branch processes the 163 attributes using a series of residual blocks. The outputs of both parallel branches are concatenated, and then passed through another series of fully connected layers.

Note that we reuse this model 3 times, once with a different pre-trained CNN.

3.3 Image CNN

As mentioned above, a pre-trained CNN processes the image and passes it to a series of linear layers. ReLU activation is used for the intermediary layers, and to avoid overfitting we a) reduce the output size by approximately half each layer and b) introduce a dropout layer right before the final layer.

Note that we use 3 different CNNs, an Inception-Resnet-V2, a Resnet-101, and a MobileNet-V2. The former was found on HuggingFace, while the other two are the Pytorch defaults. All three were pre-trained using the ImageNet database.

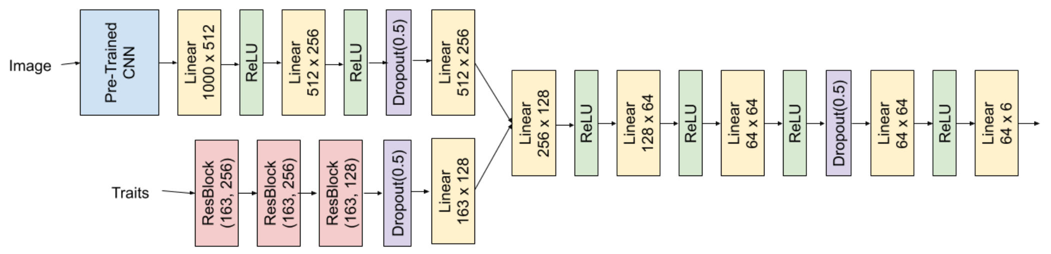


Figure 1: The layers in our base neural network. A pre-trained CNN is used to help process the image, and a parallel branch is used for the traits. Both are later combined.

3.4 Attribute Neural Net

The attribute neural net branch consists of a series of residual blocks, which are as below. The attributes are passed through a serise of layers and then added back to the output of those layers, which improves converge and allows gradients to flow more easily. The hidden layers are 256, 256, and 128 respectively, large enough to retain significant information but small enough to avoid overfitting.

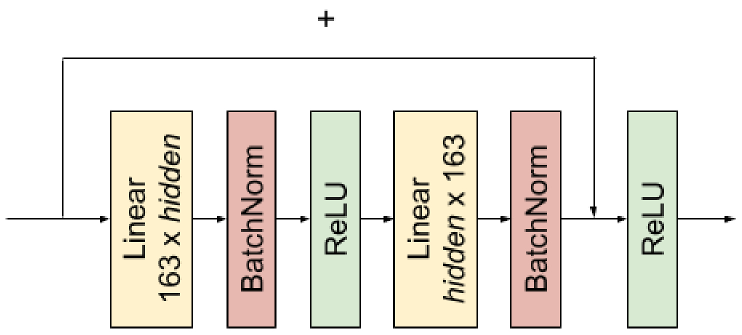


Figure 2: The layers in our base neural network. A pre-trained CNN is used to help process the image, and a parallel branch is used for the traits. Both are later combined.

3.5 Training Process

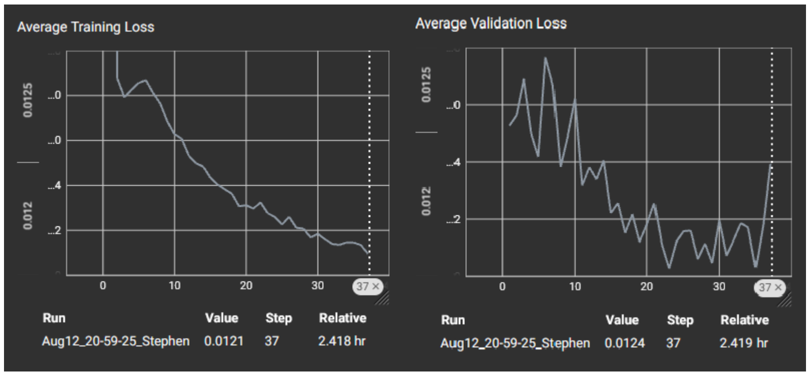
We utilize MSE loss and Pytorch's Adam optimizer in order to compute the gradients and update the model's weights using backpropagation. The optimizer's learning rate is initialized to be 0.0005 - however, model training and validation loss over time can be monitored to allow for this value to be increased or decreased based off of convergence and divergence. L2 regularization (built into the Adam optimizer) is also used to mitigate overfitting.

More formally, we are trying to minimize the following loss function. Note that practically because there are 6 attributes, loss for each attribute is averaged over the 6 traits.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1} (y_i - \hat{y}_i)^2$$

Thus by minimizing $L(y, \hat{y})$, we are simultaneously maximizing our R^2 score. To this end we look to find model parameters θ such that $\hat{y} = f(x; \theta)$ is as accurate as possible. Afterwards we compute MSE loss, and use that to compute gradients $\frac{\partial L}{\partial \theta}$ for each parameter via backpropagation. The optimizer then uses the gradients to update each parameter, and this is done repeatedly for either a set number of epochs or until a threshold minimum error has been achieved.

Each of the three models we train (i.e. using Inception-Resnet-V2, Resnet-101, and MobileNet-V2) had an R^2 score of approximately 0.2 during validation, and of approximately 0.15-0.16 on the test set. Note that the different models generally took different periods of time to train - for example the Resnet-101 model took a relatively long time (i.e. 29 epochs) to reach its peak validation R^2, after which the training and validation loss started to diverge.

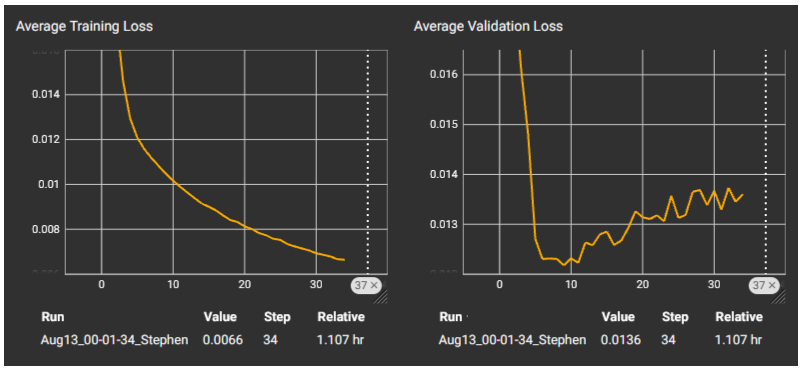


Run	Value	Step	Relative
Aug12_20-59-25_Stephen	0.0121	37	2.418 hr

Run	Value	Step	Relative
Aug12_20-59-25_Stephen	0.0124	37	2.419 hr

Figure 3: Loss over training epochs for the ResNet101 model.

In contrast, the MobileNet-V2 model's performance peaked after 11 epochs, after which training and validation loss started to diverge.



Run	Value	Step	Relative
Aug13_00-01-34_Stephen	0.0066	34	1.107 hr

Run	Value	Step	Relative
Aug13_00-01-34_Stephen	0.0136	34	1.107 hr

Figure 4: Loss over training epochs for the MobileNet-V2 model.

3.6 Ensemble of Models

We trained 3 different models using 3 different pre-trained CNNs mentioned in section 3.3. Each had an R^2 score between 0.15 and 0.16 on the test set, but when combined the score improved to above 0.19. Notably, the Inception-Resnet-V2 and MobileNet-V2 models did marginally better than ResNet101, and thus our final submission has predictions:

$$\hat{y} = 0.475\hat{y}_{\text{Inception-Resnet-V2}} + 0.475\hat{y}_{\text{MobileNet-V2}} + 0.5\hat{y}_{\text{ResNet101}}$$

4 Conclusion

Pre-trained CNNs can prove to be quite powerful and useful, as they increase the potential for the development of ground-breaking models that converge at a rapid pace. However, they also have their limitations. They can be fine-tuned very efficiently, but they also may be occassionally ill-equipped to handle the tasks at hand. Much care needs to be put into adding layers and setting hyperparameters. Novel ways to generate and use generalized pre-trained models are still being developed even now.

Ensembles also have the potential to be a force in the field, as by combining several weak outputs a stronger prediction can be made. We combined our models' outputs very simply in this project, but continuing to research more robust ways to enhance cost-efficient predictors may be well worth the effort.

References

Dong, X., Q. Wang, Q. Huang, Q. Ge, K. Zhao, X. Wu, X. Wu, L. Lei, and G. Hao (2023). “PDDD-PreTrain: A Series of Commonly Used Pre-Trained Models Support Image-Based Plant Disease Diagnosis”. *Plant Phenomics*, vol. 5, no. 0054.

Schiller, C., S. Schmidtlein, C. Boonman, A. Moreno-Martinez, and T. Kattenborn (2021). “Deep learning and citizen science enable automated plant trait predictions from photographs”. *Scientific Reports*, vol. 11, no. 16395.