Ensemble Transformer Architecture for Plant Traits Prediction

Lisa Fung
Stanford University
450 Jane Stanford Way
lisafung@stanford.edu

Annabelle Aurelia Jayadinata
Stanford University
450 Jane Stanford Way
abellej@stanford.edu

Abstract

Gathering comprehensive data on plant traits is key to understanding how plants and entire ecosystems are adapting to climate change. Currently, there is very little data on plant traits. Our goal is to predict a broad set of 6 plant traits (leaf area, plant height, specific leaf area, leaf nitrogen concentration, seed mass, and stem specific density) from crowd-sourced plant images and some ancillary data. Their traits hold the key to understanding ecosystems, e.g., in terms of their diversity, productivity, or how these plants face the challenges brought on by climate change. We applied state-of-the-art image classification architectures and developed a fine-tuned ensemble transformer architecture with the Vision Transformer and Swin Transformer for solving this problem. Using a SmoothL1Loss loss function, we achieved a final R^2 score of 0.346 on the test set. We showcase visualizations of the transformer model on a variety of input images to validate that the model is generalizing well to the plant traits for each image.

1. Introduction

From wildfires and floods to tropical storms and heatwaves, climate change is directly responsible for numerous humanitarian emergencies. Research shows that between 2030 and 2050, climate change is expected to cause approximately 250 000 additional deaths per year, from under nutrition, malaria, diarrhea and heat stress alone [1]. With the biosphere transforming at an alarming rate, plants have undergone numerous adaptations and the functioning of ecosystems have also been effected by changes in species distribution and corresponding plant trait modifications. However, predicting the global-scale impact of such phenomena is difficult to quantify due to insufficient data on plant traits. Hence, the primary objective of this project is to employ deep learning-based regression models, including Convolutional Neural Networks (CNNs) like Resnet, and Vision Transformers to predict plant traits from photographs.

Specifically, the input to our algorithm is an image of a plant (512×512 pixels). We then use CNN Architectures, Vision Transformers, and Swin Transformers with shifted windows to output 6 predicted values for the 6 plant trait measurements. Since this is a regression problem, we evaluate our model with R^2 and mean absolute error (MAE) metrics.

However, it is important to note that these plant traits, although available for each image, may not yield exceptionally high accuracies due to the inherent heterogeneity of citizen science data. Furthermore, many of the plant traits we want to predict, such as stem specific density, seed dry mass, and leaf nitrogen concentration, describe chemical tissue properties that are loosely related to the visible appearance of plants in images. Hence, we will be training a multi-modal model that can also integrate available ancillary geodata, including multitemporal optical and radar satellite data (MODIS, VOD, respectively), climate data, and soil information to supplement plant photographs, offering valuable contextual information.

1.1. Kaggle Competition

Our project serves as an entry for the Kaggle competition PlantTraits2024 - FGVC11, which aims to "advance our understanding of the global patterns of biodiversity" [5]. Kaggle is a platform that allows data scientists and machine learning engineers to find datasets, build AI models, and enter competitions to solve data science challenges [5].

2. Related Work

The field of plant trait predictions have recently undergone a transition from using purely statistical methods of prediction with aerial hyperspectral images of plants grown on plots of land to machine learning and computer vision approaches using regular plant images. Historically, Partial Least Squares Regression (PLSR), a statistical technique that finds a linear regression model by projecting observable and predicted variables to new spaces, was a popular approach for analyzing spectral data from aerial hyperspectral images that measure the optical reflectance of

plants from above using many different wavelengths [6, 8]. Then, plant trait prediction research transitioned to using traditional machine learning techniques including Random Forest, Multiple Linear Regression, Support Vector Regression, and Multilayer Perceptrons using image-based feature extraction with image analysis software and Principal Component Analysis (PCA) [7, 11]. Since the advent of Convolutional Neural Networks (CNN) [20] in computer vision and more recently, Vision Transformers [10], research in plant trait predictions from images has shifted to using CNN [27, 26, 8, 25, 19], Transformer [31], and ensemble approaches [27, 21]. These deep learning methods utilize the ability of CNNs and Transformers to detect image features that can enable improved predictions for plant traits, especially across multiple plant species.

In our work, we decided to compare Convolutional Neural Network, Vision Transfomer, and Swin Transformer [24, 23] approaches given their high demonstrated accuracies and available pre-trained models.

Past research has often focused on predicting a narrow set of one or two plant traits, often for a limited number of plant species [11, 19, 25, 3, 7, 31]. In contrast, our goal is to predict 6 plant traits simultaneously using one multi-output model, which has been shown in past research to effectively leverage correlations between plant traits [26, 27, 6, 8].

While some approaches have used image segmentation to isolate the plant image from its background [3, 19], we decided to not use segmentation because the background can provide useful contextual features such as plant environment and climate, which can better predict plant traits.

The article "Deep learning and citizen science enable automated plant trait predictions from photographs" by Schiller et al. lays the foundation for our work by combining photographs from citizen science (iNaturalist dataset) with trait observations (TRY database) based on plant species [27]. We leverage the positive results shown in [27] of using species-specific trait distributions, adding climate data, and utilizing ensemble methods.

3. Dataset

The dataset consists of over 60,000 high resolution plant images gathered from the iNaturalist database. Using the species names found in both the iNaturalist and TRY databases, trait observations obtained from the TRY database (with species-specific mean and standard deviation for each plant trait) were linked with the plant photographs (iNaturalist). Additionally, based on the location of where each photograph was taken, we also have geodatasets for each plant photograph, including temperature and precipitation data from WORLDCLIM, sand content and pH value from SOIL, optical reflectance of sunlight from MODIS satellite data, and radar constellation data from VOD, to serve as supporting information for our plant trait predictions. From this database, our goal is to predict information on 6 different plant traits, which are summarized in the table below:

trait_ID	trait_name
X4	Stem specific density (SSD) or wood density (stem dry mass per stem fresh volume)
X11	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA)
X18	Plant height
X26	Seed dry mass
X50	Leaf nitrogen (N) content per leaf area
X3112	Leaf area (in the case of compound leaves: leaf, undefined if petiole in- or excluded)

Figure 1. Visualization of sample images from the dataset.

Figure 2. Sample image with corresponding traits.

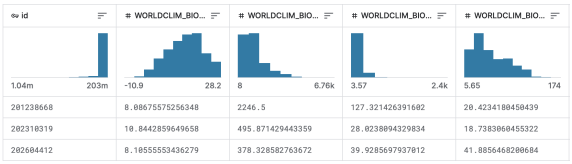

Figure 3. Visualization of geodata samples from the dataset.

3.1. Data Loader

We will be using a data loader that can simultaneously process JPEG images and tabular features as inputs. It will also be responsible for applying augmentations such as flip, random-sized crop, brightness, image compression, and normalization in batches to speed up training and reduce CPU bottleneck. Furthermore, we ran our training at various input sizes including 224×224, 256×256, and the original image size of 512×512. Our goal for these alterations is help to the model learn to generalize better to unseen data during the testing phase.

For a long time, we struggled with poor model performance, focusing solely on optimization efforts until we realized the necessity of examining our dataset. Upon closer inspection, we identified numerous outliers and observed significant variability within the plant trait data. To address these issues, we implemented specific preprocessing techniques tailored to each trait's data distribution. These techniques were applied exclusively to the training data and included:

- Filtering outliers using a quantile range (0.005, 0.995) for all traits.
- Applying a logarithmic base 10 transformation to right-skewed trait data, which included the five plant traits except trait X4 (stem specific density).
- Normalizing data to achieve a mean of 0 and standard deviation of 1 across all traits.

During inference, we apply the inverse transform, and then exponentiate the 6 outputs using base 10 to arrive at the predicted plant trait outputs. This structured approach resulted in a more optimized workflow and significantly less variability in the model's predicted plant trait outputs, significantly improving model performance during prediction.

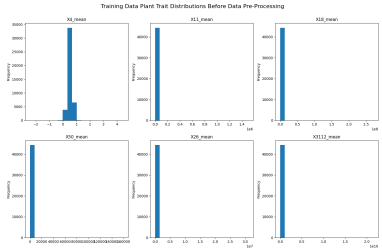

Figure 4. Training Data Plant Trait Distributions Before Data Pre-Processing

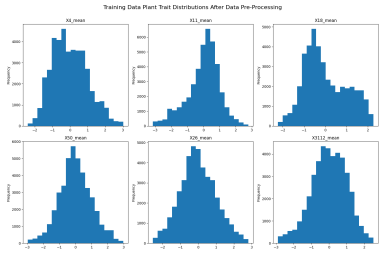

Figure 5. Training Data Plant Trait Distributions After Data Pre-Processing

3.2. Data Split

The training data set contains 55,500 plant images. We use a 4:1 split of the training set into training and validation for a good measure for validation. Specifically, we use a Stratified K Fold with k=5 to maintain similar distributions of plant traits between the training and validation sets. This leaves us with 41,797 images for the training set and 11,098 images for validation.

4. Methods

4.1. Framework and Setup

We use a Kaggle notebook, specifically utilizing Keras (TensorFlow) and PyTorch to train our network [4]. Keras offers a user-friendly framework for creating and adjusting CNN architectures. Moreover, since some of the advanced CNN architectures have readily available Keras implementations, this facilitates swift modifications to meet our project needs. Additionally, Keras simplifies image loading and resizing using OpenCV within manageable memory constraints, provides a straightforward method for loading and saving models, and allows for layer freezing. We also use PyTorch because we utilize pretrained Transformer models from Hugging Face. We run our models on Kaggle, Google Colab, and GCP with one NVIDIA T4 GPU.

4.2. Pre-trained CNN Architectures

Today, the state-of-the-art CNN architecture for feature extraction with highest-performing Imagnet classification is ResNet-50 and VGG-16.

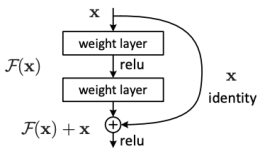

Figure 6. Residual learning: a building block. Image taken from [14]

ResNet-50 is a CNN architecture that introduces residual mappings between layers, as depicted in Figure 6, to allow for better fitting in deeper architectures. Formally, denoting the desired underlying mapping as $H(x)$, ResNet allows the stacked nonlinear layers fit another mapping of $F(x) := H(x) - x$. The original mapping is recast into $F(x) + x$. The original authors hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. In the extreme case, if an identity mapping were optimal, it would be easier to learn weights close to zero for computing $F(x) \approx 0$ than to fit an identity mapping with a stack of nonlinear layers [14]. In ResNet-50 specifically, for each residual function F, we use a stack of 3 layers which are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer as a bottleneck with smaller input/output dimensions.

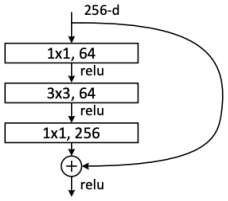

Figure 7. ResNet-50 Architecture [14]

On the other hand, the VGG-16 is a much simpler architecture consisting of 16 layers (13 convolutions and 3 fully connected layers) that is renowned for its effectiveness despite its simplicity [28]. Using transfer learning and full random initialization of the last fully connected layer, we first try these two CNN architectures on the plant trait prediction problem as they appear to be good options for feature extraction from images. Each of the two networks outputs a 3-dimensional tensor, so we used a global 2D average pooling of these CNN feature detectors and added one fully connected and dropout layer to obtain the size of our output. The resulting length-6 vector was evaluated with mean R^2 on each corresponding ground truth trait.

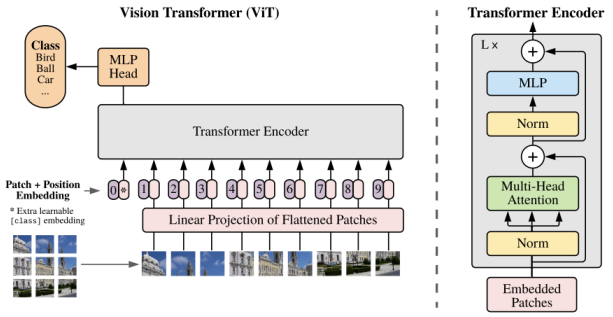

Figure 8. ViT Architecture: Images are split into fixed-size patches, linearly embedded, appended with position embeddings, and fed into a standard Transformer encoder with an extra learnable "classification token." Image taken from [10]

Furthermore, we are also interested in experimenting with pre-trained transformer models, specifically, Vision Transformer (ViT) and Swin Transformer, a vision transformer model using shifted windows. Although transformer models were firstly introduced to solve NLP (natural language processing) problems where the input is sequential data (text), it does a great job on computer vision problems tasks with multiple development in recent years.

The ViT transformer encoder (Vaswani et al., 2017) consists of alternating layers of multiheaded self-attention (MSA) and MLP blocks (Eq. 2, 3). Layernorm (LN) is applied before every block, and residual connections after every block [10]. The MLP contains two layers with a GELU non-linearity.

$$z_0 = [x_{\text{class}}; x_p^1 E; x_p^2 E; \ldots; x_p^N E] + E_{\text{pos}} \quad (1)$$

$$\text{where } E \in \mathbb{R}^{(P^2 \cdot C) \times D},\ E_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}$$

$$z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1}, \quad l = 1, \ldots, L \quad (2)$$

$$z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l, \quad l = 1, \ldots, L \quad (3)$$

$$y = \text{LN}(z_L^0) \quad (4)$$

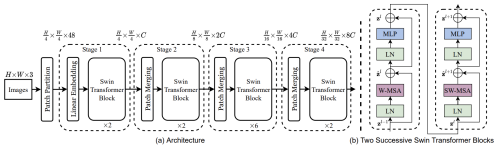

Figure 9. Swin Architecture [23]

Meanwhile, Swin Transformers rely on hierarchical feature extraction, where features are firstly extracted at different scales, and then combined to form a holistic representation of the image. At each level, features are split into non-overlapping patches and then processed by a series of transformer blocks before being fed into the next level of the hierarchy [24, 23].

Our results depicted in Table 1 showed that transformers outperformed CNNs, hence we chose to move forward with optimizing transformers as our final architecture.

4.3. Simple Architecture

To establish a baseline performance, we designed and trained a simple CNN model to predict the 6 plant trait mean values from only the images, without ancillary geodata. The architecture includes two 3x3 convolution layers, one 2x2 max pooling layer, and two fully connected layers, as depicted in 10. We use ReLU activations, batch normalization after the convolutional layers, and dropout. We trained the model using the Adam optimizer with an initial learning rate of 1e-4, step learning rate scheduler, and dropout for regularization.

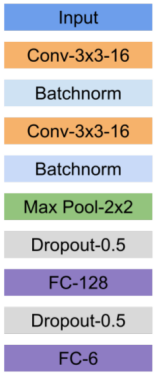

Figure 10. Simple Architecture

4.4. Evaluation & Loss Metrics

Our model will be evaluated against the independent test data provided by Kaggle. The evaluation metric for this competition is the mean R^2 (R-Squared) over all 6 traits. R^2 shows how well a regression model output (independent variable) predicts the outcome of observed data (dependent variable), and takes on values ranging from 0 to 1 (0 meaning a 0% relationship between the dependent and independent variables, and 1 indicating a 100% relationship) [2]. The equation is as follows:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where RSS = sum of squares of residuals, TSS = total sum of squares, n is the number of samples, y_i is the true value of the i^{th} sample, $\hat{y}_i$ is the predicted value of the i^{th} sample, and $\bar{y}$ is the mean of the true values.

The R^2 is commonly used for evaluating regression models and is equal to 1 minus the ratio of the sum of squares the residuals (RSS) to the total sum of squares (TSS). Since R^2 can result in large negative values, we only consider R^2 values greater than 0 for the test set. Furthermore, we will use $\frac{RSS}{TSS}$ as the loss function for the CNN models and use R^2 as the evaluation metric. For auxiliary task of predicting the standard deviations of each plant trait, where some samples don't have target labels, we will exclude them from the loss calculation using the `use_mask` argument.

For our transformer models, we found that using the SmoothL1Loss as our loss function performed better, as was used in [12]. The SmoothL1Loss is defined as

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases}$$

We selected the SmoothL1Loss because it achieves a balance between the Mean Squared Error (MSE), or L2 loss, commonly used in regression tasks, and the Mean Absolute Error, or L1 loss, which means the SmoothL1Loss is less sensitive to outliers.

Furthermore, we compute the coefficient of determination R^2 in two different ways: first, in the space after applying the $\log_{10}$ transformation (often for training), which we denote as $R^2_{\log}$, and second, in the original space before any transformation is applied, which we denote as R^2_{orig} (often for validation and inference).

5. Experimental Results

5.1. Training Details

Table 1 summarizes our various model performance metrics compared to our baseline simple CNN model depicted in 10, which performed worse than simply predicting the mean for each sample, as indicated by the negative R^2 scores.

Optimized Models	Train Loss	Train R^2	Valid Loss	Valid R^2	Test Score
Simple CNN in 10 (12 epochs)	1.1592	−0.0186	1.0570	−0.0126	N/A
Original VGG16 (12 epochs)	1.2751	0.0112	1.2649	0.0131	N/A
VGG16 w/ Pre-trained weights (12 epochs)	0.9360	0.0269	0.9253	0.04956	N/A
ResNet-50 (11 epochs)	0.9326	0.0210	0.9220	0.0481	N/A
ViT (5 epochs)	0.2695	0.4030	0.2919	0.3861	**0.286**
SwinV2 Tiny Transformer (6 epochs)	0.1964 (log)	0.5726 (log)	2827.2 (orig)	0.0402 (orig)	0.267
Swin Large Transformer (0.3 train data, 6 epochs)	0.0582 (log)	0.8812 (log)	5937.2 (orig)	0.053 (orig)	**0.277**
Final Ensemble Model	N/A	N/A	N/A	N/A	**0.346**

Table 1. Summary of Model Performance for Experiments

5.2. ResNet-50

We utilize the ResNet-50 backbone from KerasCV's pre-trained models [18] to extract features from images and Dense (fully-connected) layers to extract features from the tabular, ancillary data. We then employ two Dense layers as our final layers (heads): one without activation (for the main task) and the other with ReLU activation (for the auxiliary task). We choose ReLU for the auxiliary task because we are estimating the standard deviation of plant traits, which is always positive. Our model flow is:

- Image input → Main Task → Head
- Tabular input → Auxiliary Task → Aux Head

Note that we assign more weight to the Head than the Aux Head since Head is our main task, and our evaluation metric is calculated for the Head, not the Aux Head.

Furthermore, a well-structured learning rate schedule is essential for efficient model training, ensuring optimal convergence and avoiding issues such as overshooting or stagnation, which we have implemented as shown in Figure 12.

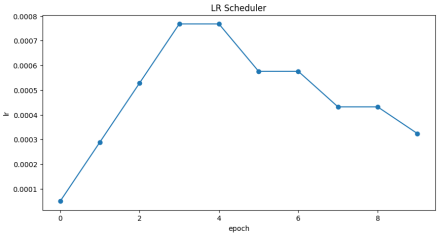

Figure 12. Learning Rate Scheduler

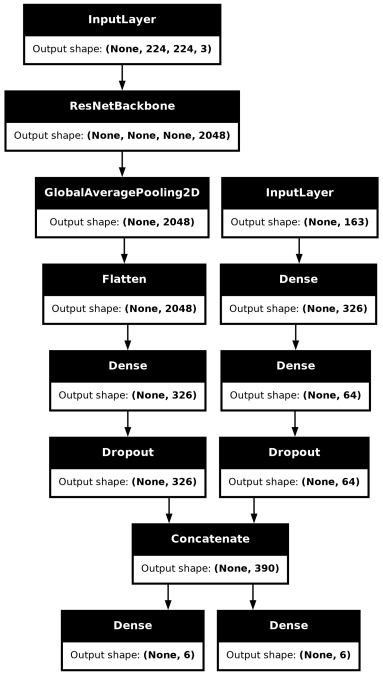

Figure 11. Architecture with ResNet-50 Backbone

Best Val R^2_{orig}	Best Epoch	Best Val loss	# Parameters (Trainable/Total)
0.0481	7	0.9220	$24,255,090/24,308,210$

Table 2. ResNet-50 Training Results

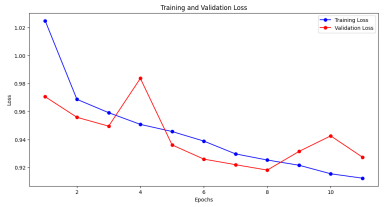

Figure 13. Training (blue) vs. Validation (red) Loss for ResNet-50

After extensive hyperparameter tuning, including adjustments to layers, dropouts, and experimenting with freezing and unfreezing techniques in ResNet-50, we found that these changes did not lead to significant improvements. Consequently, we decided to explore an alternative backbone architecture and proceeded to evaluate VGG-16, which is discussed in the following section.

5.3. VGG-16

We employed analogous data preprocessing, optimization methods, and a learning rate scheduler during training with both a custom layer-by-layer implementation of the VGG-16 backbone as well as with pre-trained weights [30]. Despite significantly increasing trainable parameters to better align with the training dataset, improvements remained limited. After extensive attempts with various other pre-trained backbones, we encountered persistent challenges. Hence, we made a strategic shift to adopt transformer architectures, prompted by discussions in subsequent lectures and insights from guest speakers.

Best Val R^2_{orig}	Best Epoch	Best Val loss	# Parameters (Trainable/Total)
0.0496	11	0.925	$119,884,004/134,598,692$

Table 3. VGG-16 Training Results

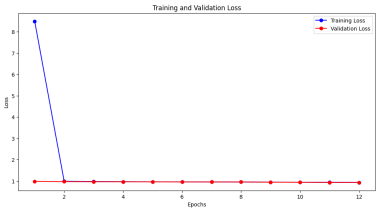

Figure 14. Training (blue) vs. Validation (red) Loss for VGG-16

5.4. Vision Transformer

Utilizing a pre-trained Vision Transformer (ViT) model [10], such as those available from sources like Google's version on Hugging Face [33] [9], has enabled achieving impressive training R^2_{log} scores approaching 0.8-0.9, within a small number of epochs (fewer than 10). This initial success underscores the ViT model's ability to capture complex patterns in the training data effectively.

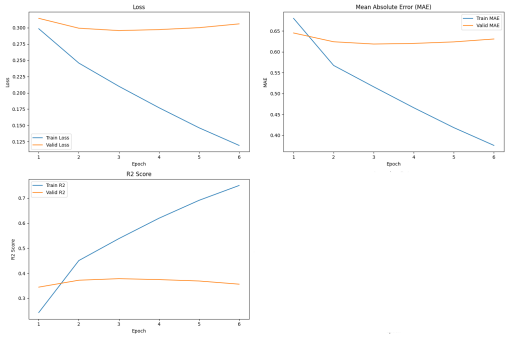

Figure 15. Training (blue) vs. Validation (orange) Loss for ViT V1

However, despite achieving high R^2_{log} scores during training, the model demonstrates significant overfitting when evaluated on unseen data, leading to markedly worse performance with R^2 values below 0.05 for inference submissions. To mitigate these challenges, several key strategies were implemented:

- Weight Decay Application: Utilizing the AdamW optimizer with a `weight_decay` parameter set to 0.01 ensures effective L2 regularization. This regularization helps prevent the model from relying too much on specific features of the training data, thereby improving its generalization ability to unseen data.
- Dropout Regularization: Incorporating dropout layers within the model architecture introduces stochasticity

during training, effectively reducing overfitting by randomly dropping units and encouraging the network to learn more robust features.

- Early Stopping Mechanism: Implementing an early stopping strategy based on validation loss monitoring helps prevent the model from training excessively and overfitting by halting training once the validation loss ceases to improve.
- Batch Normalization: Integrating batch normalization layers normalizes the input to each layer, stabilizing learning and accelerating convergence during training. This technique enhances the model's ability to generalize by reducing internal covariate shift [16].

The ViTForRegression model architecture exemplifies these strategies, where batch normalization is applied to the output of the ViT model's classification token. This normalization step, coupled with dropout regularization and early stopping, collectively contributes to enhancing the model's performance and mitigating overfitting issues. In fact, in the early stages of training, the validation loss is lower than the training loss due to such regularization techniques and training data augmentation.

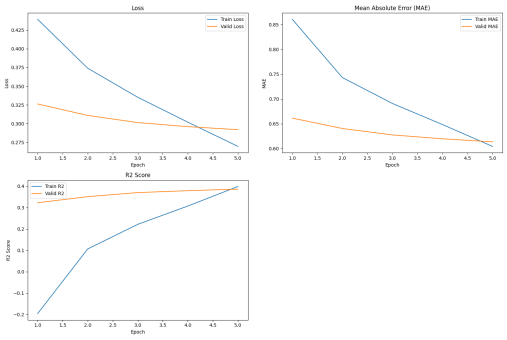

Figure 16. Training (blue) vs. Validation (orange) Loss for ViT V2

Best Val R2	Best Epoch	Best Val loss
0.3258	5	0.3159

Table 4. ViT V2 Training Results

5.5. Swin Transformer

Inspired by Kaggle user HDJoJo [13], we test the Swin Transformer, a hierarchical Vision Transformer that uses shifted windows to limit self-attention to local windows and enable cross-window connections [24, 23].

Specifically, we fine-tune three different pre-trained Swin Transformer models from Hugging Face that take in the plant images as input and output 6 plant trait values:

1. SwinV2 Tiny Window Size 16x16, Image Size 256x256 [23, 32]
    - 28.3M parameters, Pretrained on ImageNet-1k
2. SwinV2 Small Window Size 16x16, Image Size 256x256 [23, 32]
    - 49.7M parameters, Pretrained on ImageNet-1k
3. [24, 32]Swin Large Window Size 12x12, Image Size 384x384
    - 196.7M parameters, Pretrained on ImageNet-22k and Fine-tuned on ImageNet-1k

With the SwinV2 Tiny pre-trained model, we fine-tune 3 different models. We train our baseline SwinV2 Tiny model with the 1cycle learning rate policy [29] with maximum learning rate 1e-4, batch size of 10, for 6 epochs. After 6 epochs, it produced a training $R^2_{\text{log}} = 0.5726$, validation $R^2_{\text{orig}} = 0.0402$, and test $R^2_{\text{orig}} = 0.26691$. Since the validation R^2_{orig} score was increasing across all epochs for the baseline model, we decided to train the same model for longer and with a larger batch size. Thus, we train a second SwinV2 Tiny model with the same 1cycle learning rate policy, batch size of 16, for 12 epochs. The resulting model displayed even more overfitting as it reached a training $R^2_{\text{log}} = 0.7448$ and validation $R^2_{\text{orig}} = 0.0367$ after 12 epochs. The second SwinV2 Tiny with increased batch size achieved the best validation $R^2_{\text{orig}} = 0.03954$ after 8 epochs, and performed with $R^2_{\text{orig}} = 0.24936$ on the test set.

Next, we experiment with integrating the $\log_{10}$ transform directly into the Swin Transformer model instead of using it in the data preprocessing. To do so, we add a final layer that performs a 10^x operation for all five outputs x corresponding to the plant traits $X11, X18, X26, X50, X3112$ (excluding trait $X4$). We calculate the loss and R^2 in the original space instead of the $\log_{10}$ transform space. For this experiment, we only apply the filtering outlier step for data preprocessing. We train the log-adjusted SwinV2 Tiny model with the same 1cycle learning rate policy, batch size of 10, for 12 epochs. This resulted in a training $R^2_{\text{orig}} = 0.0795$, validation $R^2_{\text{orig}} = 0.0188$, and testing $R^2_{\text{orig}} = 0.12825$. The log-adjusted Swin Transformer V2 Tiny model clearly underperforms the previous SwinV2 Tiny models, which demonstrates how Swin Transformer models perform best with normalized and more symmetrically distributed outputs that we can produce with our data preprocessing steps.

Using the SwinV2 Small pre-trained model, we fine-tune one model. The SwinV2 Small model is trained with the same 1cycle learning rate scheduler, batch size of 10, for 6 epochs. This model achieved a training $R^2_{\text{log}} = 0.6593$, validation $R^2_{\text{log}} = 0.3640$, and testing $R^2_{\text{orig}} = 0.27308$. The training statistics of loss, R^2_{log}, mean absolute error (MAE), and learning rate are shown in Figure 17. There is clear overfitting as the training loss and MAE decrease more rapidly than the validation loss and MAE, and the training R^2_{log} increases more rapidly than the validation R^2_{log}.

We see that the larger SwinV2 Small model performs better than the SwinV2 Tiny models, even though both display clear overfitting. Thus, so we decide to test a larger

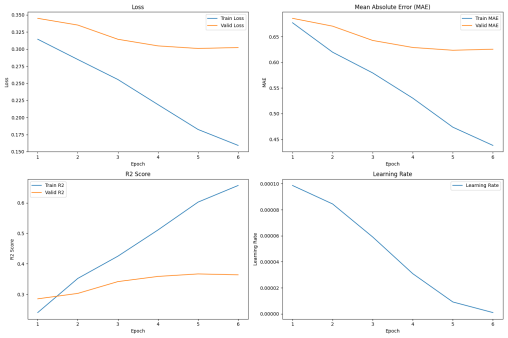

Figure 17. SwinV2 Small Model Training (blue) vs. Validation (orange) Statistics of Loss, $R^2_{\log}$, MAE, and Learning Rate

Swin Transformer Model	Train R^2		Validation R^2		Test R^2
	$R^2_{\log}$	R^2_{orig}	$R^2_{\log}$	R^2_{orig}	R^2_{orig}
SwinV2 Tiny baseline	0.5726	X	X	0.0402	**0.26691**
SwinV2 Tiny second (12 epochs)	0.7448	X	X	0.0367	0.21284
SwinV2 Tiny second (8 epochs)	0.6317	X	X	0.03954	0.24936
SwinV2 Tiny log-adjusted	X	0.0795	X	0.0188	0.12825
SwinV2 Small	0.6593	X	0.364	X	**0.27308**
Swin Large	0.8812	X	X	0.053	**0.27745**

Table 5. Summary of Swin Transformer Models Performance

Swin Transformer model with more dropout layers.

Using the Swin Large pre-trained model, we train it with the same 1cycle learning rate policy with maximum learning rate 1e-4, batch size of 10, for 6 epochs, with a random 0.3 sample of the training data due to memory and runtime constraints. This achieved a training $R^2_{\log} = 0.8812$, validation $R^2_{\text{orig}} = 0.0530$, and test $R^2_{\text{orig}} = 0.27745$.

After training the Swin Transformer models, we visualize the self-attention saliency maps. We select the SwinV2 Tiny baseline model for visualization because it has the simplest transformer architecture, which means the first Window Attention layer has a larger contribution to the resulting plant trait scores, and because it performed the best of the SwinV2 Tiny models. We display the self-attention maps of the first Window Attention layer for the validation set with corresponding R^2 scores for each plant trait, as well as for the test set in Figures 18 and 19.

5.6. Final Ensemble Architecture: Swin Large and ViTForRegression

We applied ensemble modeling to combine our best models, the Swin Large Transformer (test $R^2 = 0.27745$) and ViTForRegression (test $R^2 = 0.28584$). By averaging their predictions, we achieved a final test R^2 score of 0.34612, indicating that the ensemble successfully harnessed the complementary feature representations and predictive strengths of both models, improving our prediction accuracy compared to using either model alone.

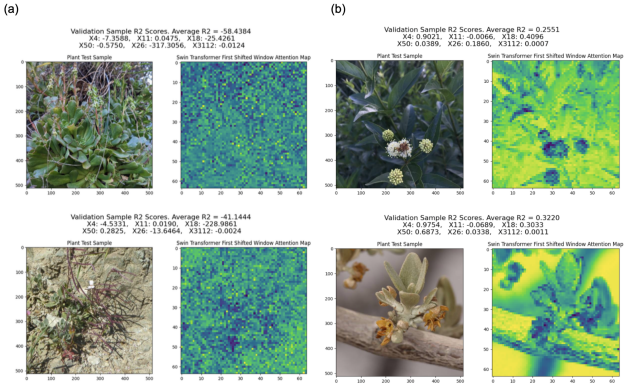

Figure 18. SwinV2 Tiny baseline Validation Attention Maps. (a) Poor Attention (b) Good Attention

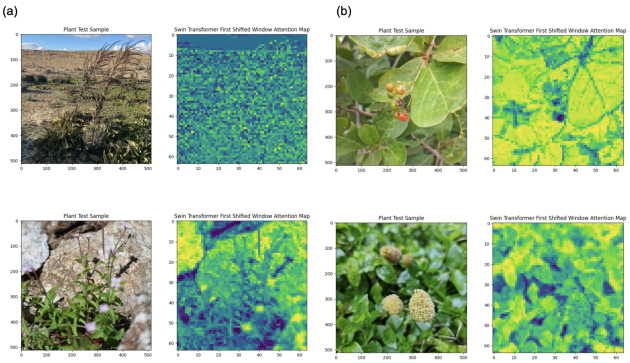

Figure 19. SwinV2 Tiny baseline Test Attention Maps. (a) Poor Attention Maps (b) Good Attention

6. Conclusion & Future Work

We explored various algorithms for predicting plant traits from images, including CNN models like VGG16 and ResNet-50, as well as vision transformer models like ViT and Swin Transformer. While traditional CNN models struggled to achieve satisfactory performance, transformer models, especially pre-trained ViT and Swin Transformer, showed promising results. This discrepancy in performance can be attributed to transformers' ability to effectively capture long-range dependencies in images, which is crucial for understanding complex visual patterns in plant traits.

Moving forward, further investigation into techniques like LoRA [15] or DoRA [22] for parameter-efficient fine-tuning could enhance model performance, particularly in resource-constrained environments in order to make transfer learning for specific tasks more effective and precise. Additionally, exploring regularization methods such as the SMART algorithm [17] may help improve the robustness and generalization capabilities of the models. These avenues of research could lead to even more accurate and efficient plant trait prediction models.

7. Contributions & Acknowledgements

Annabelle conducted experiments with ResNet-50, VGG-16, and ViT models, while Lisa conducted the data processing, implemented and evaluated the baseline CNN and Swin Transformer models, and created the attention map visualizations. Both authors contributed to writing the paper. We thank the author of the Kaggle dataset for publicly providing labeled plant images and their corresponding traits, as well as @HDJoJo (Kaggle) for providing some insight into using Swin Transformers for plant trait prediction. Last but not the least, we are thankful for all the CS231N teaching staff for this wonderful class!

References

[1] Climate change, 2023. www.who.int/news-room/fact-sheets/detail/climate-change-and-health. 1

[2] I. V. Abba. What is r squared? r2 value meaning and definition. https://www.freecodecamp.org/news/what-is-r-squared-r2-value-meaning-and-definition/: :text=An 5

[3] S. Aich, A. Josuttes, I. Ovsyannikov, K. Strueby, I. Ahmed, H. S. Duddu, C. Pozniak, S. Shirtliffe, and I. Stavness. Deepwheat: Estimating phenotypic traits from crop images with deep learning. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 323–332, 2018. 2

[4] G. P. C. A. C. f. Awsaf, Martin Görner. Planttraits2024: Kerascv starter notebook. https://www.kaggle.com/code/awsaf49/planttraits2024-kerascv-starter-notebook, 2024. Accessed: April 6, 2024. 3

[5] H.-J. i. M. G. T. K. Awsaf, AyushiSharma. Planttraits2024 - fgvc11, 2024. 1

[6] A. Capolupo, L. Kooistra, C. Berendonk, L. Boccia, and J. Suomalainen. Estimating plant traits of grasslands from uav-acquired hyperspectral images: A comparison of statistical approaches. *ISPRS International Journal of Geo-Information*, 4(4):2792–2820, 2015. 2

[7] D. Chen, R. Shi, J.-M. Pape, K. Neumann, D. Arend, A. Graner, M. Chen, and C. Klukas. Predicting plant biomass accumulation from image-derived parameters. *GigaScience*, 7(2):giy001, 01 2018. 2

[8] E. Cherif, H. Feilhauer, K. Berger, P. D. Dao, M. Ewald, T. B. Hank, Y. He, K. R. Kovach, B. Lu, P. A. Townsend, and T. Kattenborn. From spectra to plant functional traits: Transferable multi-trait models from heterogeneous and sparse data. *Remote Sensing of Environment*, 292:113580, 2023. 2

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 2, 4, 6

[11] N. T. Duc, A. Ramlal, A. Rajendran, D. Raju, S. K. Lal, S. Kumar, R. N. Sahoo, and V. Chinnusamy. Image-based phenotyping of seed architectural traits and prediction of seed weight using machine learning models in soybean. *Frontiers in Plant Science*, 14, 2023. 2

[12] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 5

[13] HDJoJo. Modified planttraits2024: Eda+training. https://www.kaggle.com/code/hdjojo/modified-planttraits2024-eda-training, 2024. Accessed: May 13, 2024. 7

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. 3, 4

[15] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 8

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 7

[17] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. 8

[18] Keras. Libraries. https://keras.io/api/keras_cv/models/backbones/resnet_v1/. 5

[19] S. Kolhar and J. Jagtap. Plant trait estimation and classification studies in plant phenotyping using machine vision – a review. *Information Processing in Agriculture*, 10(1):114–135, 2023. 2

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 2

[21] C. P. Lee, K. M. Lim, Y. X. Song, and A. Alqahtani. Plant-cnn-vit: Plant classification with ensemble of convolutional neural networks and vision transformer. *Plants*, 12(14), 2023. 2

[22] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. Dora: Weight-decomposed low-rank adaptation, 2024. 8

[23] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo. Swin transformer v2: Scaling up capacity and resolution. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2, 4, 7

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2, 4, 7

[25] M. P. Pound, J. A. Atkinson, A. J. Townsend, M. H. Wilson, M. Griffiths, A. S. Jackson, A. Bulat, G. Tzimiropoulos, D. M. Wells, E. H. Murchie, T. P. Pridmore, and A. P.

French. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *GigaScience*, 6(10):gix083, 08 2017. 2

[26] P. Raja, A. Olenskyj, H. Kamangir, and M. Earles. Simultaneously predicting multiple plant traits from multiple sensors via deformable cnn regression, 2021. 2

[27] C. Schiller, S. Schmidtlein, C. Boonman, A. Moreno-Martínez, and T. Kattenborn. Deep learning and citizen science enable automated plant trait predictions from photographs. *Scientific Reports*, 11(1):16395, Aug. 2021. Publisher: Nature Publishing Group. 2

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 4

[29] L. N. Smith and N. Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017. 7

[30] VGG16. Pytorch libraries. https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html. 6

[31] Y. Wang, C. Wang, B. Wang, and H. Wang. Combination of feature selection methods and lightweight transformer model for estimating the canopy water content of alpine shrub using spectral data. *Infrared Physics Technology*, 139:105304, 2024. 2

[32] R. Wightman. Pytorch image models. https://github.com/huggingface/pytorch-image-models, 2019. 7

[33] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda. Visual transformers: Token-based image representation and processing for computer vision, 2020. 6