Final Project Report

Tracy Zhu, Bale Chen

May 2023

1 Task formulation

In this project, we attempt to solve a plant seedling image [2] classification task[1]. In particular, given an image of a crop seedling, we are going to develop a method that automatically determines which species of the 12 kinds of crops each image is from. The objective is to achieve a high micro F1-score on the test set.

2 Data

Investigation We first look into the class distribution of the training dataset (See Figure 1). The majority class Loose Silky-bent has twice as many datapoints as the 5 minority classes on the right of the graph, indicating a clear imbalance in the train set. Train set consists of 4750 samples while the test set has 792 samples.

With a closer look at the real samples, we found that black grass and loose silky-bent seedlings are nearly identical to lay-people's eyes (See Figure 2a and 2b).

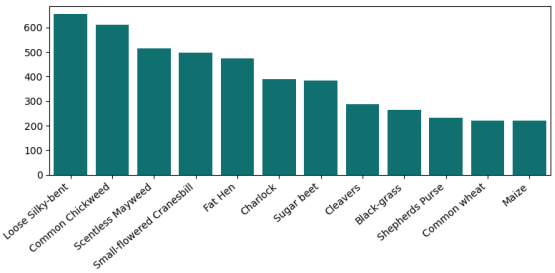
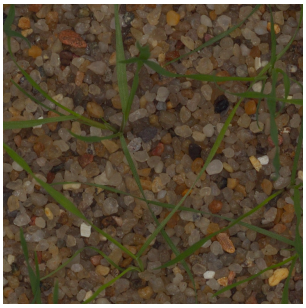


Figure 1: Number of images in each class

Preprocessing Since the images are not of uniform sizes, we first prepossessed the images to 224x224 using random cropping. Then, we normalized the input images tensors using the fixed mean and variance from ImageNet, since we are going to perform transfer learning using models pretrained on ImageNet.

(a) Black Grass



(b) Loose Silky-bent

Figure 2: Sample Images

[1]The task is originally from this kaggle competition: https://www.kaggle.com/competitions/plant-seedlings-classification

Data augmentation Through a few baseline experiments, we found that models are not learning data from the minority classes well, especially for the Black Grass class where the best-performing-models remain under 0.8 validation accuracy. We decided to apply more complicated data augmentations to help model learn, other than basic resizing and normalization. Through a few common sense on plant images and empirical evaluations, we decided to adopt the following augmentations:

- Random cropping.
- Change in directions, including Horizontal Flipping, Vertical Flipping, and Random Rotations with degree between -150 and 150 degrees.
- Gaussian Blur to erase noise in the data since the data are real-world photographs.
- Change in colors, including Random Grayscale to turn images into black-and-white color and Color Jitter with parameters [brightness=0.3, contrast=0.2, saturation=0.2, and hue=0.2].
- Eliminating a part of information in an image, including Random Erasing.

Since shapes can be important features for the model to distinguish between different classes, we did not adopt transformations that alter shapes of objects in the image such as Elastic Transform. An example image with the above augmentations is shown in Appendix A.

Dataset oversampling Additionally, we tried to address the problem of dataset imbalance by oversampling with varying augmentations. Specifically, first we oversample each of classes to the size of the largest class in the imbalanced dataset. This sampling also roughly doubles the size of the imbalanced dataset. In the oversampling process, we tried varying augmentations: each image gets assigned an augmentation from 3 augmentation sampled uniformly from the list of augmentations described before, so that one image can have different augmentations when it is sampled more than once. Unfortunately all our experiment results showed this strategy led to models overfitting the datset. Therefore we did not pursue this approach in further experiments.

3 Model architecture

3.1 Pretrained models

We have adopted three popular classes of pretrained models for our image classification task. Our selection was based on two main factors: the models' generalizability and model complexity (measured by the number of trainable parameters). Additionally, we considered computational efficiency and excluded models that have too many trainable parameters and would take a long time to train,. We ended up in selecting VGG, the ResNet family, and ViT.

For each of these models, we have replaced their final linear layer with an output dimension of 12. By leveraging these powerful pre-trained models and fine-tuning them for our speciic classification task, we aim to achieve good model performance and efficient training.

VGG VGG[5] is one of the winners in 2014 ImageNet Challenge. We adopt VGG16 with a total of 138 million trainable parameters and 16 weight layers. VGG utilizes a deep architecture with a large number of trainable parameters and small filters, which enable it to learn increasingly complex and abstract features from images. Using small filters also reduces the number of parameters in each layer, which makes it possible to use a deeper architecture without overfitting the training data.

ResNet The ResNet[3] model, introduced in 2015, is one of the most widely used models. Resdopting residual connections help address the problem of vanishing gradients that can occur in very deep netowrks. The idea behind residual connections is to add shortcut connections that bypass one or more layers of the network, allowing the input to flow through the network more easily. We tried different sizes of ResNet as an investigation on model complexity, including ResNet18 (11.7 million trainable parameters), ResNet50 (25.6 million), ResNet101 (44.5 million), ResNet152 (60.2 million), wide ResNet50 (68.9 million), and wide ResNet101 (126.9 million).

ViT Vision Transformer[1] is an attention-based model introduced in 2020 with 86.6 million trainable parameters. Unlike CNNs, which rely on convolutional and pooling operations to extract local features, ViT divides the input image into a set of non-overlapping patches and applies a self-attention mechanism to capture the dependencies between these patches. ViT also employs a transformer architecture, which is widely used in natural language processing tasks, to process the image patches.

3.2 Ensemble

After using fine-tuned individual models, we also experimented with ensemble method. Particularly the stacking method is adopted, which leverages the output logit vectors from multiple models to generate final prediction. The detailed precedure is shown in Figure 3. M pretrained models are first fine-tuned independently on the train set. We then pass the train data through the fine-tuned models to generate M logit vectors $\{\hat{y}_i^{(j)}\}_{i=1}^{M}$ for each datapoint $j = 1, 2, \cdots, N$. We concatenate all the logit vectors for each data point to form a long vector $\tilde{y}^{(j)} = \hat{y}_1^{(j)} \oplus \hat{y}_2^{(j)} \oplus \cdots \oplus \hat{y}_M^{(j)}$. Lastly, we train a logistic regression on $\mathcal{D} = \{\tilde{y}^{(j)}, y^{(j)}\}_{j=1}^{N}$.

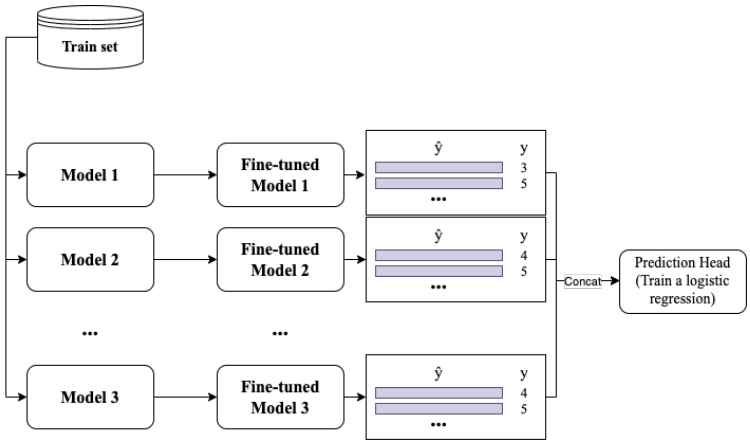


Figure 3: Flowchart of our ensemble method

The logistic regression is thus a prediction head of the whole ensemble. The rationale behind stacking these models is that instead of using argmax function to generate final prediction, the logistic regression learns how to generate better predictions based on the logit values from different models. In other words, if model A is better at predicting some class than the other models, the logistic regression learns to rely on logits from model A to predict labels for that class.

4 Methods

4.1 Hyperparameter search

We used the RayTune [4] library to perform hyperparameter search . It has multiple convenient APIs for organizing automatic tuning sessions. In our implementation, learning rate and batch size are the two hyperparameter that needs to be tuned. We first used the loguniform search for learning rate in range of $[10^{-5}, 10^{-2}]$ and grid search for batch size in $\{16, 32, 64, 128\}$. The loguniform search is perfect for exploring values across different orders of magnitudes, while the grid search can exhaustively test all listed batch sizes. With this coarse hyperparameter space, we are able to narrow down the best performing learning rate range $[10^{-5}, 2 \times 10^{-4}]$ and batch sizes $\{32, 64\}$. In later hyperparameter searches, we used the smaller space to conduct more efficient hyperparameter searches.

4.2 Learning rate scheduler

In tuning the learning rate as explained in 4.1, we found that inappropriate choices of learning rates often result in models getting stuck at undesirable levels of performance such as validation accuracy = 0.6 or 0.8. We suspect that models may get stuck at local minima or saddle points in training. Therefore we decided to adopt a Cyclical learning rate scheduler [6] to help models find a global minimum in the loss landscape. A cyclical learning rate scheduler involves setting a range of learning rates that are cycled between during training, as opposed to monotonically reducing the learning rate, as shown in Figure 4.

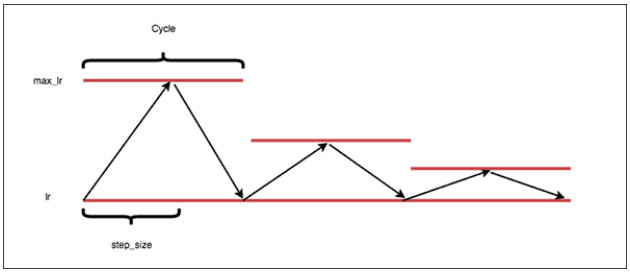


Figure 4: Cyclic Learning Rate Scheduler

After a series of empirical evaluations, we chose base_lr=1e-5 and max_lr=1e-2.

5 Results & Discussion

Phase 1 We first applied transfer learning the pretrained VGG16, ResNet50, and Vision Transformer as baseline results. Learning rate was fixed at 10^{-4} and batch size was set to be 32. Seeing from the training and validation loss curve (Figure 5), the baseline models all converged within 4 epochs. The test set F1-score is recorded at the first row of Table 1.

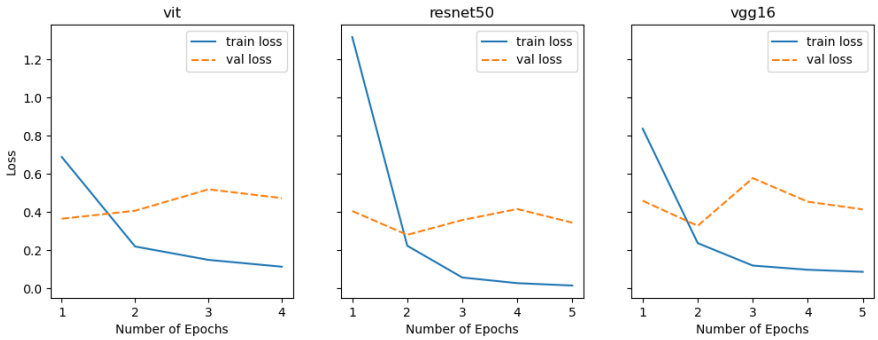


Figure 5: Training and Validation Loss Curve of Three Baseline models

To improve on the baseline, we first tuned the hyperparameters. With the tuning technique mentioned in Section 4.1, we found the best performing hyperparameters based on validation loss. The hyperparameters and test set F1-score can be found in Table 1.

Phase 2 After achieving a test score of 0.97229, we found that more hyperparameter tuning had marginal improvement. The per class accuracy graph of ResNet50 showed that the model is handling most classes near perfectly except for the black grass class. As we demonstrated in Section 2, black grass is very similar to loose silky-bent which happens to have the most samples in the training set. An error analysis showed that of all the black grass images that are classified wrongly, 98% were classified as loose silky-bent.

	VGG16	**ResNet50**	**ViT**
Baseline Test F1	0.93576	0.95717	0.93073
Best Learning Rate	2.45×10^{-5}	3.58×10^{-5}	1.07×10^{-5}
Best Batch Size	32	32	64
Tuned Test F1	0.95591	0.97229	0.96473

Table 1: Baseline and Hyperparameter Tuning Results

We then leveraged data augmentation, learning rate scheduler, and model scaling to help model better learn the minority class features. With data augmentation, we found the training process takes more epochs to converge, and the accuracy of black grass class is steadily increasing (See Figure 8).This indicates that adding difficulty with data augmentation to the training process potentially help model learn the more useful features of the images, leading the model to better distinguish the black grass class. We also tried to augment the dataset by oversample the minority classes with `WeightedRandomSampler` from PyTorch, but this method leads to significant overfit.

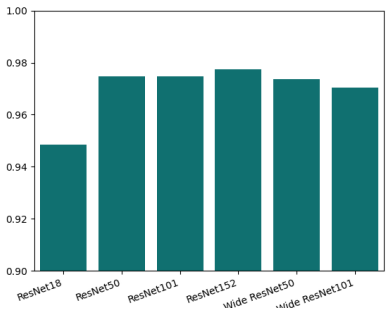


Figure 6: The relationship between model size and test set F1-score

Regarding the relationship between model size and performance, we investigated this using the different sizes of ResNet models. In Figure 6, we see that up until ResNet152, the test set performance is positively correlated with model size. Yet, due to potential overparameterization or architectual difference, the wide ResNet models have degraded test score.

Method	**Test F1-score**
Fixed LR + ResNet152 + Data Aug	0.97858
Cyclic LR + ResNet101 + Data Aug	0.97481
Ensemble Method	**0.98362***

Table 2: Results of the Phase 2 Methods

Since we have experimented multiple configurations of different models, we are convinced that training one end-to-end model is reaching a bottleneck. Thus, we implemented the ensemble method with ResNet 50, 101, 152, and VGG16, which eventually achieved our best score 0.98364 on the test set. Table 2 showed the three final methods we tried in Phase 2. The ensemble method clearly broke through the bottleneck of one-model methods. This result suggests that models with different sizes and architectures shows varying capability in this multi-class classification task, where ensemble method is able to learn those advantages of different models.

Limitation and Future Work First, our implementation of oversampling method does not contribute to handling the imbalance in this dataset. There are two plausible reasons: we oversampled too many images, or our strategy of uniformly sampling three augmentations did not produce a complicated sample.

Additionally, since eventually the validation accuracy of black grass class is still significantly lower than other classes, a better ensemble framework might help discriminate black grass against loose silky bent. For example, we can train a separate binary classification model to predict whether an image is black grass or loose silky-bent seedling. When the multi-class classification model has low confidence in predicting those two classes, we use the binary classification model to generate a potentially more accurate label.

References

[1] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[2] Thomas Mosgaard Giselsson et al. "A public image database for benchmark of plant seedling classification algorithms". In: *arXiv preprint arXiv:1711.05458* (2017).

[3] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[4] Richard Liaw et al. "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118* (2018).

[5] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[6] Leslie N Smith. "Cyclical learning rates for training neural networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472.

Appendices

A Example image after data augmentation

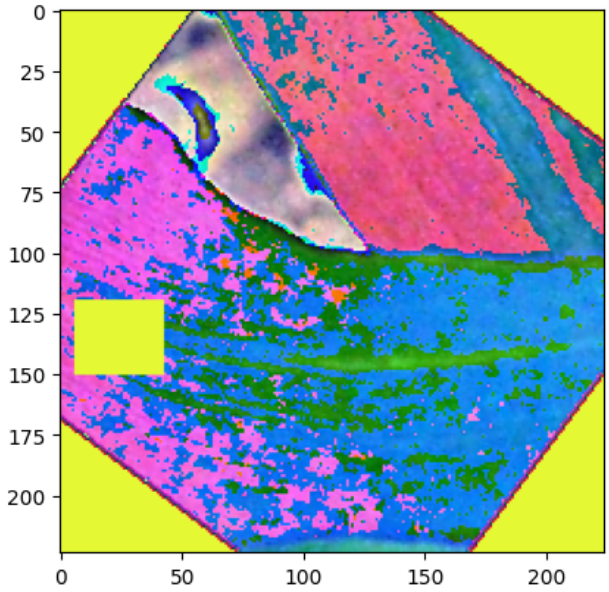


Figure 7: Example image after data augmentation

B Per Class Accuracy Before and After Data Augmentation

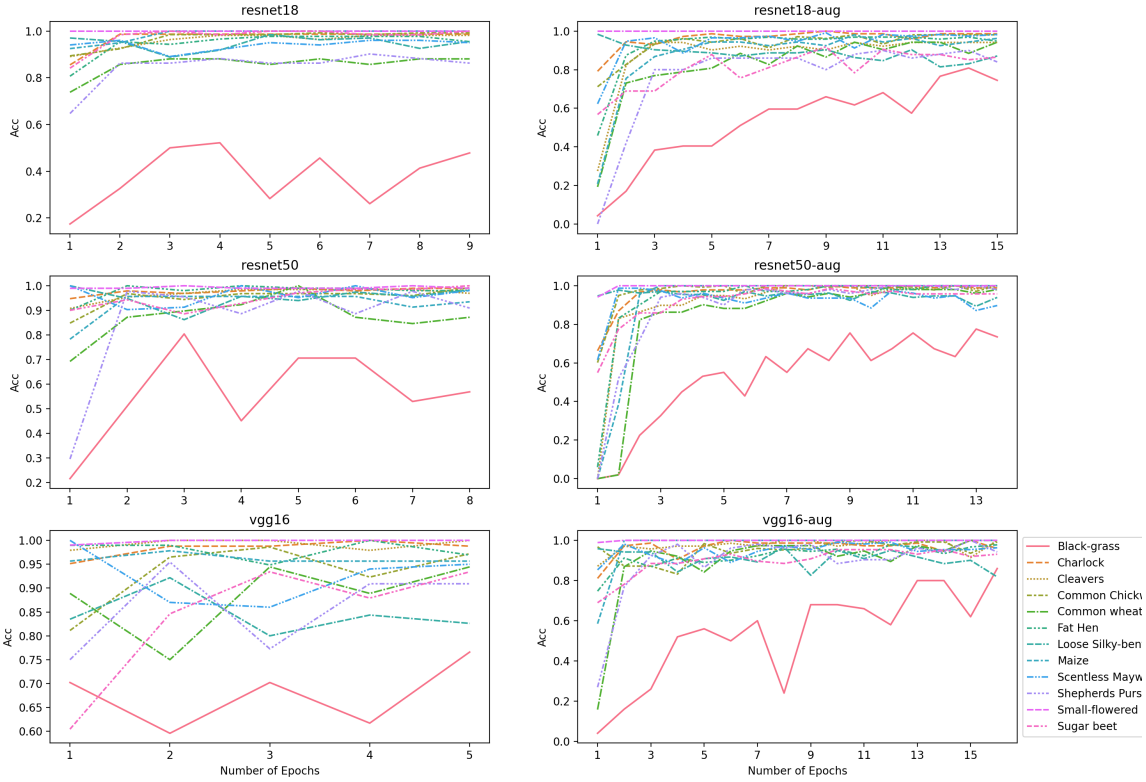


Figure 8: Per Class Accuracy Before and After Data Augmentation