Predicting Vital Plant Traits With ResNet50 and Regression Model

Howe Wang
School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
h884wang@uwaterloo.ca
report due: August 12

Abstract

This project aims to predict certain features of plants, given their respective images and ancillary information. Using ∼43000 observation data acquired by iNaturalist observations, the model is trained by first classifying using ResNet50 to output an initial prediction of 6 features, then combined with ancillary data to predict a more accurate prediction of 6 features. The resulting network has an R2 score of ∼0.14 on never seen before datasets. The training code is in this github link.

1 Introduction

CNN Networks are extremely efficient and powerful in processing and classifying large amounts of images, and can be used to replace tedious human classification in many cases. One of such cases is the prediction of plant traits. In the 2021 paper by Christopher Schiller and others, they described the necessity of correctly classifying these information: "Plant functional traits (‘traits’) are essential for assessing biodiversity and ecosystem processes, but cumbersome to measure." (Schiller, 1)

This paper aims at providing a solution to this prediction by utilizing ResNet50’s efficient image processing power, as well as using a fully connected two layer Linear Model to provide accurate traits predictions by utilizing all data. The workflow is demonstrated in Figure 1.

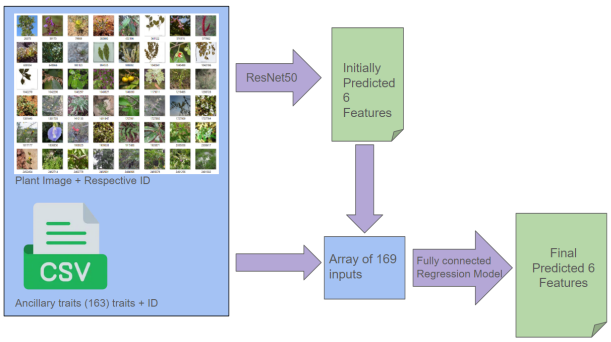

Figure 1: Workflow of training process. The Plant Images were first processed using ResNet50 to predict features, the features are then combined with ancillary traits to form a more informed prediction

Submitted to CS480/680, 2024 Spring. Do not distribute.

2 Related Works

Similar approaches have been considered in the past. In Schiller's paper, which utilizes an ensemble of CNNs to train the network, the CNNs were also given the bio-climate data to make more informed processing. Schiller reported an significant increase in R2 score when considering bio-climate information. The main difference comparing Schiller's approach and the approach described in this paper is that Schiller's approach passes the data in through the CNN, while the approach explored in this paper passes that information after the CNN gives its estimate.

3 Main Results

Some of the important design decisions made in this paper are:

1. Normalizing data. Normalizing the input ancillary information and output labels normalizes the weights of the inputs and outputs, allowing the Regression Model to train more accurately without influences of traits with high numeric values. Similarly, normalizing the input images (R, G, B matrices) provides consistent input range and improves convergence.

2. Breaking the training process into image training and numeric training. Not only does this allow for training using pretrained models without much augmentation, it also allows seamless swapping of different parts of the training process to experiment for better results, and demonstrates clearly the improvements by doing so. as shown in Figure 2, the running comparison between the different training algorithms provide good insights into the benefits of choosing a combined model to run.

	Mock R2 Score during Testing	R2 Score from Kaggle
ResNet50 Only	0.12468	0.12365
Regression Only	0.0738	N/A
ResNet50 With 3 Layer Regression	0.44637	0.13684
ResNet50 With 2 Layer Regression	0.46812	0.14732

Figure 2: Results collected from different training processes. Note that using Regression with ancillary data provided significant improvement to ResNet50's results, while 2 Layer Regression performs better than 3 Layers. This shows the benefits of choosing to run with both ResNet and Regression.

3. Breaking the training dataset into training and validation sets. By using a small part of the training dataset to validate scoring after each epoch can prevent over-fitting and provide the most realistic R2 scoring possible.

4. Saving the best model after each epoch. This prevents over-fitting by always keeping track of the current best predicting model based on the validation dataset. This also prevents computer issues and crashes which may arise.

Ratinoale/Algorithm used in training:

ResNet50 and Regression Model were chosen for training because of their efficiency of training time to training results. ResNet requires two and a half hours to run before it starts overfitting, while the regression model takes 30 minutes to run. This makes them the perfect choice as they are able to provide a good result within a set amount of time.

The following Figure 3 shows the helper algorithms and classes used in the training algorithm.

The training is broken up into two parts, the image training part and regression training part. Figure 4

```
class PlantDataset:
      // Processes each data into its own dataset
      returns image, label, ID

// Regression Model used
class RegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(RegressionModel, self).__init__()
        self.layer1 = nn.Linear(input_dim, 64)
        self.layer2 = nn.Linear(64, 32)
        self.output = nn.Linear(32, output_dim)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = self.output(x)
        return x

// Evaluates the current model. r2_score function already defined
Algorithm: evaluate_model
Input: model, dataloader
      true_labels = []
      predictions = []
      for input, label in dataloader:
            output = model(input)
            append label to true_labels
            append output to predictions
      return r2_score(true_labels, predictions)

// Predicts the imformation and saves to an csv
Algorithm: predict_and_save
Input: model, dataloader, output_file
      predictions = []
      for inputs, labels, id in dataloader:
            output = model(inputs)
            renormalize output
            append (id, output) to predictions
      convert to dataframe
      save dataframe as output_file
```

Figure 3: **PlantDataset** class is used for formatting data, while **evaluate model** and **predict and save** are used to calculate metrics and save predictions. The Regression Model used is also defined

```
// Actual training of algorithm (ResNet50)
Algorithm: Resnet Training Algorithm
Input: train_csv, train_images, test_csv, test_images
      // Normalize
      normalize ancillary information
      normalize Y labels
      normalize images

      // Define and pre-process all data and model used
      define ResNet50
      change ResNet50's # of output to 6
      Convert all data into PlantDataset
      Break Train's data into train and validation
      Load all PlantDatasets into dataloader

      // Training
      define num_epoches
      best_model = model
      best_R2 = -inf
      for epoch in num_epoches:
            for input, label in train_loader:
                  predict output
                  calculate loss
                  loss.backward()
                  optimizer.step()
            current_R2 = evaluate_model(model, val_loader)
            if currentR2 > best_R2
                  best_R2 = current_R2
                  best_model = model

      // Post-process
      predict_and_save(best_model, val_loader, output_file)
      Add ResNet's predictions into train_csv
      Add ResNet's predictions into test_csv
```

```
// Actual training of algorithm (Regression)
Algorithm: Regression Training Algorithm
      // Normalize
      normalize ancillary information
      normalize Y labels

      // Define and pre-process all data and model used
      define Regression Model
      Break Train's data into train and validation
      load data into dataloader

      // Actual Training
      define num_epoches
      best_model = model
      best_R2 = -inf
      for epoch in num_epoches:
            for input, label in train_loader:
                  predict output
                  calculate loss
                  loss.backward()
                  optimizer.step()
            current_R2 = evaluate_model(model, val_loader)
            if currentR2 > best_R2
                  best_R2 = current_R2
                  best_model = model

      predict_and_save(best_model, val_loader, output_file)
```

Figure 4: **PlantDataset** class is used for formatting data, while **evaluate model** and **predict and save** are used to calculate metrics and save predictions. The Regression Model used is also defined

Results During Training:

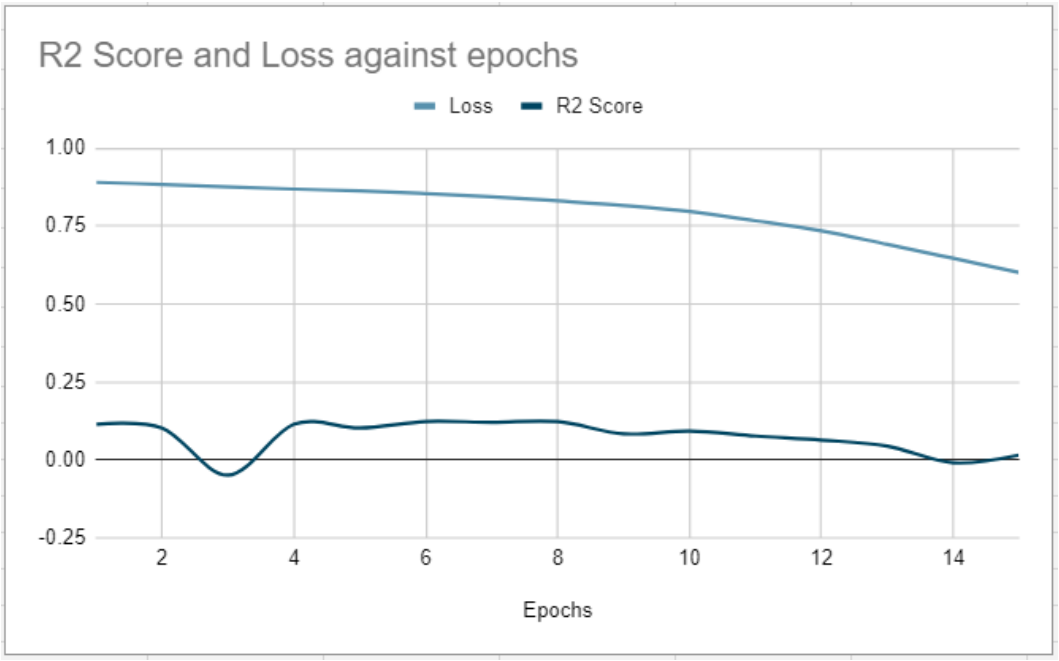

Figure 5: Figure shows the Loss and R2 score during ResNet's training. It can be seen to start overfitting around 13 epochs in.

4 Conclusion

Through the training process proposed in this paper, we learnt that both the image and ancillary information contributes to predicting features of the plant. This model is limited by ResNet's ability to discern plant features, and swapping ResNet50 with other neural networks can result in better performance. For example, in paper "Deep-plant: Plant identification with convolutional neural networks", customly defined CNNs fitted to discern features of plants can be better suited for this task, and using an ensemble of CNNs can provide better results, though would take significantly longer to train.

While this training model is not fully utilizing and tailored to the subject of the dataset, it is easy to swap components and provides a strong foundation for training datasets with both images and numeric values.

Acknowledgement

Thanks to Saber Malekmohammadi for helping and providing suggestions to the training process.

References

Lee, S. H., C. S. Chan, P. Wilkin, and P. Remagnino (2015). "Deep-plant: Plant identification with convolutional neural networks". In: *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 452–456.

Oquab, M. et al. (2024). "DINOv2: Learning Robust Visual Features without Supervision". arXiv: 2304.07193 [cs.CV].

Schiller, C., S. Schmidtlein, C. Boonman, A. Moreno-Martínez, and T. Kattenborn (Aug. 2021). "Deep learning and citizen science enable automated plant trait predictions from photographs".