CS 480 2024 Project Report

Jeannie Quach
School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
`jeannie.quach@uwaterloo.ca`
report due: August 12

Abstract

This project focuses on predicting plant traits from citizen science plant photographs, and their corresponding ancillary information, as part of the final project of CS 480. The goal was to reach a baseline of 0.43902 and I achieved a score of 0.49471 publicly. To achieve this score, I utilized advanced machine learning techniques such as using the DINOv2 Vision Transformer model (giant model) for feature extraction and XGBoost for regression analysis. Detailed documentation is available in the GitHub repository: here.

1 Introduction

Initially, I faced poor results with a custom model, leading me to explore pretrained models. I found DINOv2 (Vision Transformer), a self-supervised model effective for image classification, object detection, and video understanding. DINOv2 learns from visual data without labeled examples, making it robust across datasets. I used DINOv2_vitg14_reg, with 1.1 billion parameters, which achieved 81.5% classification accuracy on iNaturalist, proving it suitable for extracting image embeddings. The following 1 outlines the steps from data preprocessing to model evaluation.

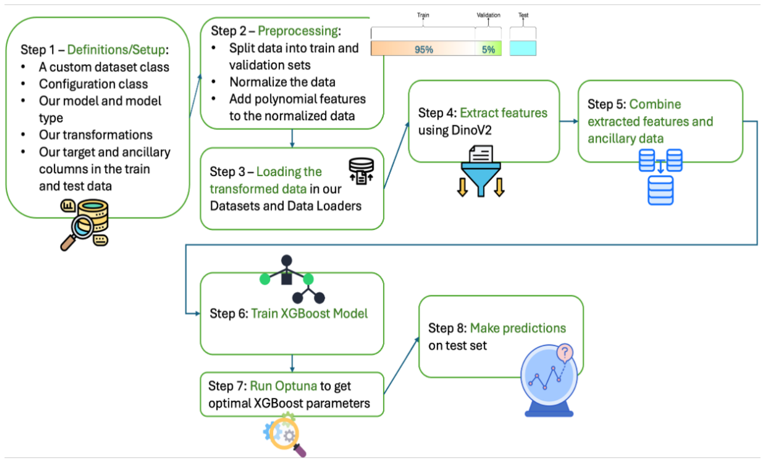
Step 1 – Definitions/Setup:
• A custom dataset class
• Configuration class
• Our model and model type
• Our transformations
• Our target and ancillary columns in the train and test data
Step 2 – Preprocessing:
• Split data into train and validation sets
• Normalize the data
• Add polynomial features to the normalized data
Train
Validation
Test
95%
5%
Step 3 – Loading the transformed data in our Datasets and Data Loaders
Step 4: Extract features using DinoV2
Step 5: Combine extracted features and ancillary data
Step 6: Train XGBoost Model
Step 7: Run Optuna to get optimal XGBoost parameters
Step 8: Make predictions on test set

Figure 1: Steps taken to solve this problem

2 Related Works

Predicting plant traits from images has gained attention recently. An earlier study demonstrated the use of Convolutional Neural Networks (CNNs) for this task, and a Kaggle competition in June 2024 also focused on this, with participants using various techniques similar to mine. The key difference from the Kaggle competition is the lack of standard deviation data in the CS480 dataset.

Aleksandr Korotaevskiy used a similar approach in the Kaggle competition, including DINOv2 for feature extraction. However, Korotaevskiy cleaned data by removing extreme values and managed data differently, using an extra path in the CSV for images.

3 Main Results

To predict plant traits from images and ancillary data, my approach involved several key steps:

1. Creating a custom dataset class to store the training and test data columns along with corresponding JPEG images, and applying defined transformations to the images.
2. Defining a pretrained model (DINOv2) to extract image embeddings or targeted features.
3. Reading the CSV files and defining target and ancillary data columns.
4. Splitting the data into training and validation sets (95% train, 5% test).
5. Scaling and normalizing the data using Standard Scaler.
6. Applying polynomial feature transformation to capture non-linear relationships between features and target traits.
7. Loading the training and validation data and images into the custom dataset class.
8. Extracting image embeddings using the model and dataset. Combining the features with scaled ancillary data for model training.
9. Training six XGBoost models—one for each trait—using the numerical data and image embeddings.
10. Using the trained models to predict the target traits in the test dataset.
11. Submitting the prediction results.

The following sections detail each component and design choice, supported by ablation studies where relevant.

3.1 Custom Dataset Creation

Following PyTorch best practices, I implemented a custom dataset class to efficiently manage both image and numerical data. It's important to note that on previous assignments, we have used PyTorch datasets and data loaders to handle large datasets, such as the MNIST dataset from Torchvision. This approach aligns with PyTorch's recommendation to separate data processing from model training for better readability and maintainability. By integrating images and numerical data in this custom dataset, consistent transformations were applied, as guided by this Pytorch tutorial. The following class is my custom dataset:

3.2 Model Creation and Selection

Initially, I faced poor results with a custom model and transformations, which led me to explore pretrained models. I discovered DINOv2 (Vision Transformer), a self-supervised model effective for image classification, object detection, and video understanding.

DINOv2 excels in learning representations from visual data without labeled data, making it robust across various datasets. I selected DINOv2_vitg14_reg, which has 1.1 billion parameters and has demonstrated strong performance, including an 81.5% classification accuracy on the iNaturalist dataset.

3.3 Preprocessing data

To prepare the data for training, I employed standard preprocessing techniques such as train-test splitting and scaling/normalization using Standard Scaler.

3.3.1 Train-Test Split and K-Fold Cross-Validation

To split my dataset, I used `train_test_split` for its simplicity and reproducibility via `random_state`, which was useful for debugging and comparing different DINOv2 models.

After testing various splits, a 95% training and 5% testing ratio yielded the best results. Figure 2 shows the model's performance with different split ratios, models, and transformations without polynomial features being applied. All models used the same transformations unless custom ones from Hugging Face were applied. I also tested 5-fold cross-validation, which achieved an R2 score of 0.3846. However, the 95-5 split with the DINOv2-giant model provided the best balance of performance and efficiency.

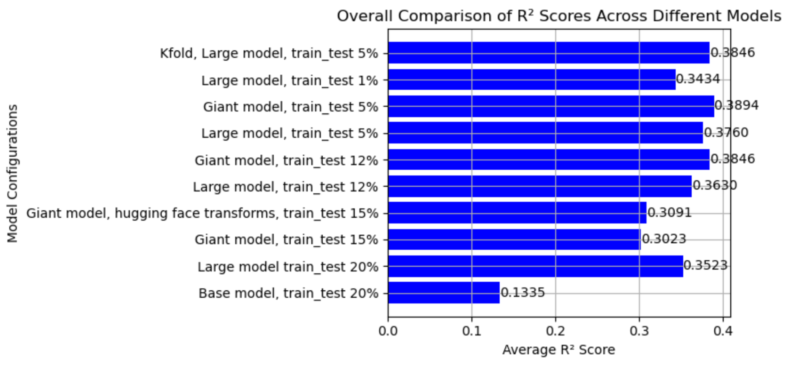

Figure 2: Overall comparison of R2 scores between different models and train, test sizes

3.3.2 Transformations

The transformations were applied based on the recommended settings in the DINOv2 documentation file documentation file. The following transformations were employed: resize, center crop, conversion to tensor, and normalization. These transformations are designed to maximize the pretrained model's effectiveness by ensuring consistency with the data it was originally trained on.

3.3.3 Polynomial Features

Polynomial features capture complex, non-linear relationships and interactions between variables, which improved my model's performance. By adding polynomial features, the R2 score on Kaggle increased from 0.49175 to 0.49471. However, this improvement came at the cost of increased computational time, with predictions taking about 5 hours instead of 40 minutes. An overall comparison of the R2 scores with and without polynomial features can been seen in Figure 3

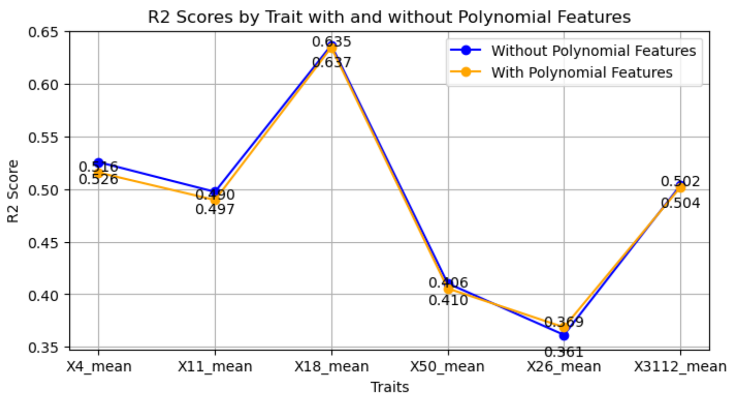

Figure 3: R2 scores by trait with and without polynomial features

3.4 Loading Model and Extraction image method

I experimented with two methods for loading the DINOv2 model: torch.hub.load (PyTorch) and AutoImageProcessor (Hugging Face). While the PyTorch method was more flexible and easier to use, the Hugging Face method was quicker in extracting image embeddings. However, due to its limitations in applying custom transformations and accessing specific DINOv2 models, I ultimately opted for PyTorch.

3.5 CatBoost vs XGBoost

CatBoost and XGBoost are both gradient boosting algorithms for predictions based on embeddings. After reading this article, I found XGBoost to be more suitable for handling missing values and mixed data types, offering greater flexibility and control. XGBoost's popularity, with 19.3 million downloads last month compared to CatBoost's 2.25 million, also reinforced my choice.

3.6 Hyperparameter Tuning with Optuna

To find the best XGBoost parameters, I initially trained the model with default settings and experimented manually, which was time-consuming. I then used Optuna, a hyperparameter optimization tool, following a tutorial. Optuna helped me find optimal parameters through 50 trials, taking about 24 hours. Figure 4 shows the process of finding the optimal parameters.

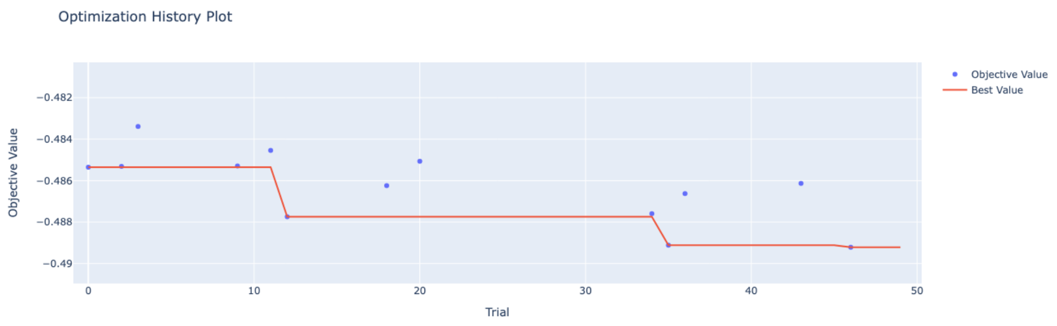

Figure 4: R2 scores over 50 trials

4 Conclusion

This project deepened my understanding of machine learning, particularly in model selection, feature engineering, and the practical use of tools like PyTorch and XGBoost. I learned the importance of careful hyperparameter tuning and the impact of techniques like cross-validation on model performance. This project took me roughly around a week and a half to complete to my desired score and if it were not for other final projects and exams, I would have liked to continue improving my score. In the future, I would like to look into model stacking, feature interaction and alternative models.

Acknowledgement

I would like to thank Professor Yao-Liang Yu and the CS480 TAs for organizing the Kaggle competition and final project. Special thanks to Srihari Vishnu for helping me fix a critical bug and deepening my understanding, and to Erik Lungulescu for his valuable guidance and insights into key machine learning frameworks.

References

"CatBoost" (2024).

"Data Loading and Processing Tutorial" (2024).

Iljin (2021). "Comparing the Titans of Machine Learning: XGBoost, CatBoost, and LightGBM".

Korotas (2024). "9th Place Plant Traits 2024: DINOv2 + CatBoost".

"Plant Traits 2024" (2024).

Research, F. (2024a). "Evaluation".

– (2024b). "Transforms.py".

Wang, H. and et al. (2021). "A Comprehensive Evaluation of XGBoost for Time Series Forecasting". *Scientific Reports*, vol. 11, no. 1.

"XGBoost Hyperparameter Tuning with Optuna" (2023).