Analysis of Leafsnap Dataset

Introduction

The Leafsnap dataset was developed by researchers from the University of Washington, Columbia University, the University of Maryland, and the Smithsonian Institution, funded by the National Science Foundation and Google. The purpose was to create an "Electronic Field Guide" for identifying tree species with just a picture of its leaves. This came in the form of an iPhone app that was released in May 2011. Later, A paper on the subject was published in Computer Vision - ECCV 2012. The paper talks about how the researchers used a nearest neighbors classification for determining which leaf is in the picture [Kumar et al., 2012]. I suspect that with a more complex model, such as neural networks, I could replicate and possibly improve the app's results. Using the Leafsnap dataset, I trained various types of neural networks, like convolutional neural networks (CNN), MobileNetV2, and ResNet50. For faster model runtime, I also chose species with the most pictures of them. After various attempts at making neural networks and waiting hours for them to train, I concluded that a simpler model, such as the one used in the paper, is better for the task. Not to say that a neural network cannot solve this problem with higher efficiency, I just do not have the computational power and expertise in fine-tuning necessary for doing so. After I struggled with neural networks, I created my own KNN to solve this problem and had more success. However, while cleaning the data differently for my KNN, I decided to reapply it to my neural networks, and the accuracy greatly improved.

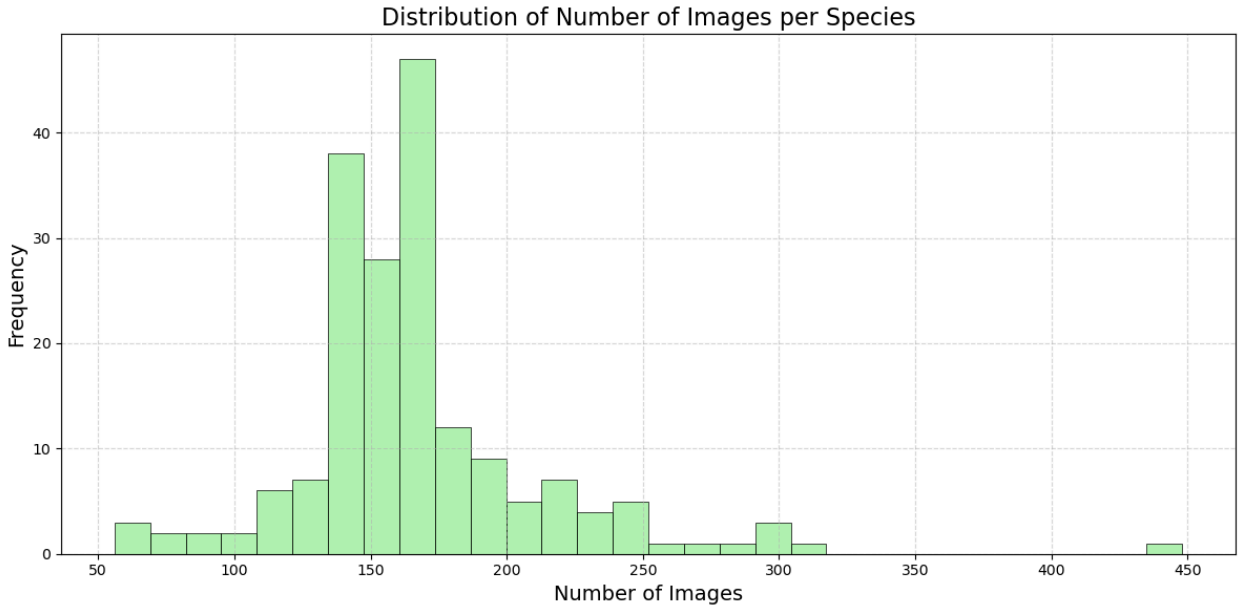
Methods

The primary dataset that I used for the project was the official Leafsnap dataset. I approached this problem multiple ways, through segmentation and breaking the image down to various sizes for better model digestion. The goal was for the model to accurately predict which species of tree a given leaf was from. I preprocessed the data in a few different ways. Because this dataset is so massive, it took a long time to run my neural networks on it, so I needed to make it smaller. The first way I approached this was by looking at the number of images per species on a histogram. I chose to keep the top ten species with the most images, focusing on images taken in a lab setting. The lab pictures had a color calibration plot and were more consistent, so this helped with training the model. Later in my testing, I noticed that some of the segmented lab images were completely black or only displayed the color calibration card. To combat this problem, I created a function that looked at images with a majority of black pixels and threw out a species if a majority of their pictures were bad. This worked well for my KNN and basic CNN models. However, when I applied data augmentation, dynamic learning rate, and transfer learning models like MobileNetV2, there was severe overfitting and other issues during model training. I needed to give my MobileNetV2 more complex data than just black and white segmented images. I input the color images instead, and the accuracy drastically increased. With some more fine-tuning of hyperparameters, I was able to reach a validation accuracy of 98%.

Order of Operations

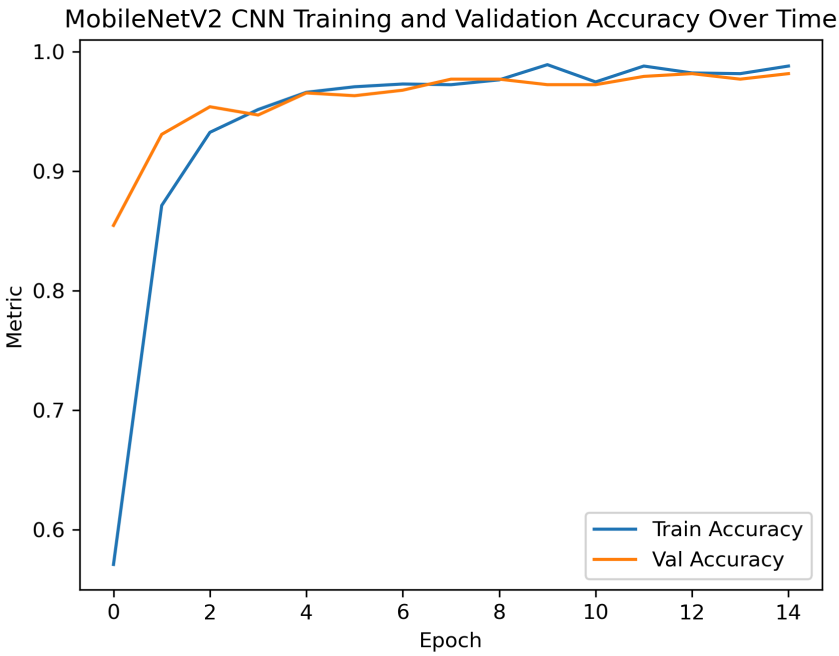
1. Create a dataset containing only the top ten species by the number of images taken in the lab
2. Split the data into training and testing sets

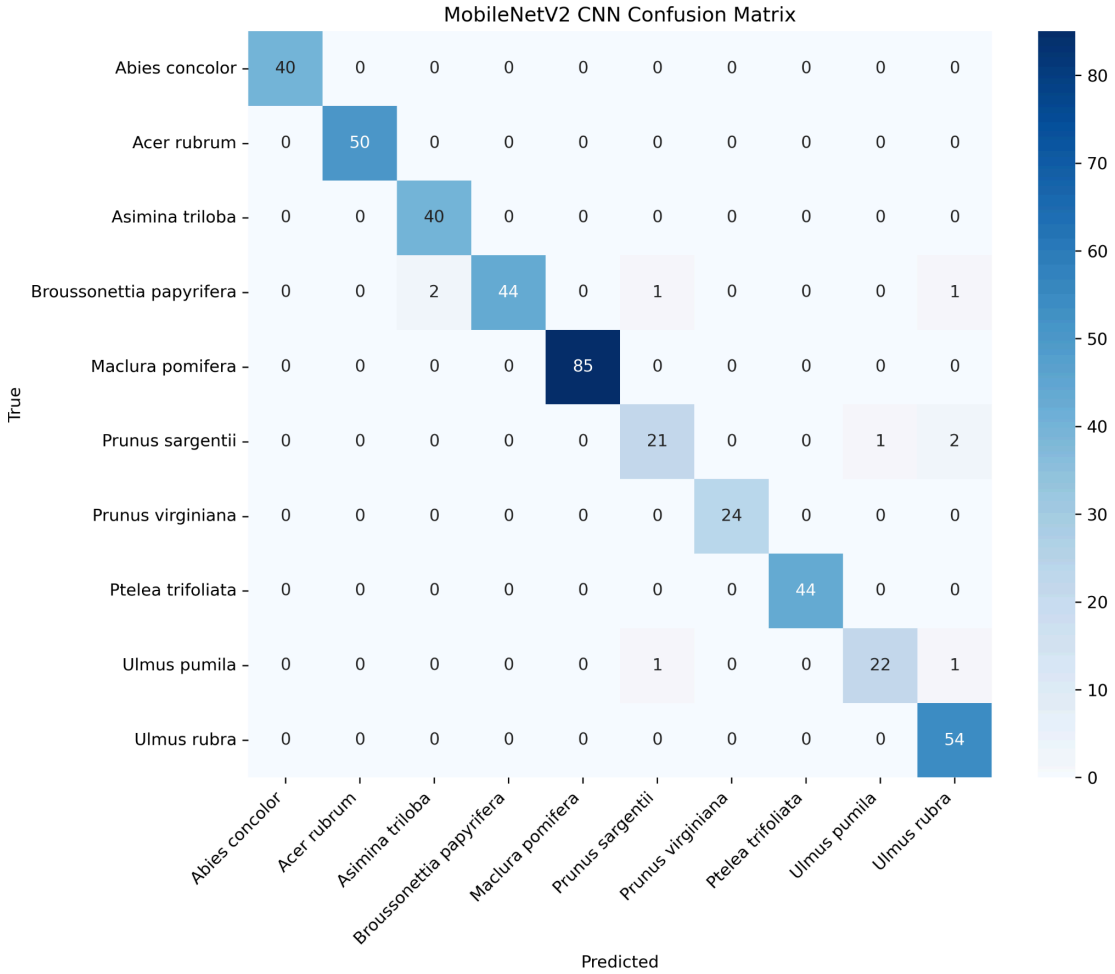
3. Create TensorFlow datasets from the split data
4. Produce a data augmentation pipeline
5. Apply data augmentation to the training data
6. Shuffle and batch the data
7. Load MobileNetV2 pretrained base model
8. Build and compile the full model
9. Define early stopping and dynamic learning rate
10. Train the model
11. Evaluate and visualize



Results

My initial analysis of the dataset with neural networks did not have very high accuracy. However, once I updated my data preprocessing, the accuracy rose greatly. I plotted the training and validation accuracy over fifteen epochs. Additionally, I made a confusion matrix from the output of my MobileNetV2 CNN to display which species were getting misclassified.


MobileNetV2 CNN Training and Validation Accuracy Over Time
Metric
Epoch
Train Accuracy
Val Accuracy
1.0
0.9
0.8
0.7
0.6
0
2
4
6
8
10
12
14


MobileNetV2 CNN Confusion Matrix
True
Predicted
Abies concolor
Acer rubrum
Asimina triloba
Broussonettia papyrifera
Maclura pomifera
Prunus sargentii
Prunus virginiana
Ptelea trifoliata
Ulmus pumila
Ulmus rubra
40 0 0 0 0 0 0 0 0 0
0 50 0 0 0 0 0 0 0 0
0 0 40 0 0 0 0 0 0 0
0 0 2 44 0 1 0 0 0 1
0 0 0 0 85 0 0 0 0 0
0 0 0 0 0 21 0 0 1 2
0 0 0 0 0 0 24 0 0 0
0 0 0 0 0 0 0 44 0 0
0 0 0 0 0 1 0 0 22 1
0 0 0 0 0 0 0 0 0 54
80
70
60
50
40
30
20
10
0

Discussion

The training and validation plot for the MobileNetV2 CNN model showcases the model's good performance. There is a steep curve in the beginning, signifying that the model improves fastest during the first few epochs. After the training validation line reaches around the 0.9 mark, the validation line follows, showing that there is no significant overfitting. Both curves slowly converge as the epochs progress close to the 0.98 mark. The epoch parameter was set equal to twenty, but the early stop mechanism stopped it at fifteen epochs. The learning rate was also reduced closer as the lines converged as not to skip over the best balance of parameter weights.

The confusion matrix for the MobileNetV2 CNN model shows strong classification performance and near-perfect classification among the ten classes. The class Maclura pomifera contains the most images and therefore has the most members in the validation set, which shows zero misclassifications. Despite the model's high accuracy, a few small misclassifications occurred when identifying a Broussonettia papyrifera as an Asimina triloba. These misclassifications are likely due to visual similarities.

When interpreting the results of my model, it could be used by a researcher, an arborist, or someone whose hobby is tree identification. The model and this project were mainly constructed as a proof of concept and an educational exploration of CNNs. The project was more work than it could have been, but I am glad I stuck with it because I learned a lot.

I encountered many hurdles during this project, namely, poor organization in the beginning. I was too excited to jump right into the new concept of convolutional neural networks and apply them to this massive dataset that I did not do a very good job of cleaning the data before processing. When I became frustrated with the lack of performance in my neural networks, I changed gears to a KNN model similar to the one used in the Leafsnap paper [Kumar et al., 2012]. When I experienced success with my KNN model, I went back to neural networks with my newly cleaned dataset. This yielded me great success, and I'm glad I took the time to go back to it. This project has inspired me to continue working with CNN and other image processing algorithms because it is very satisfying to me and can be a fun side project.

Work Cited

Kumar, N., Belhumeur, P. N., Biswas, A., Jacobs, D. W., Kress, W. J., Lopez, I. C., & Soares, J. V. B. (2012). Leafsnap: A computer vision system for automatic plant species identification. In Computer Vision – ECCV 2012 (pp. 502–516). Springer.

Hu, X. (2019). *Leafsnap Dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/xhlulu/leafsnap-dataset

The link on the official Leafsnap site gave a 404 (Page Not Found), so I had to download it from Kaggle.