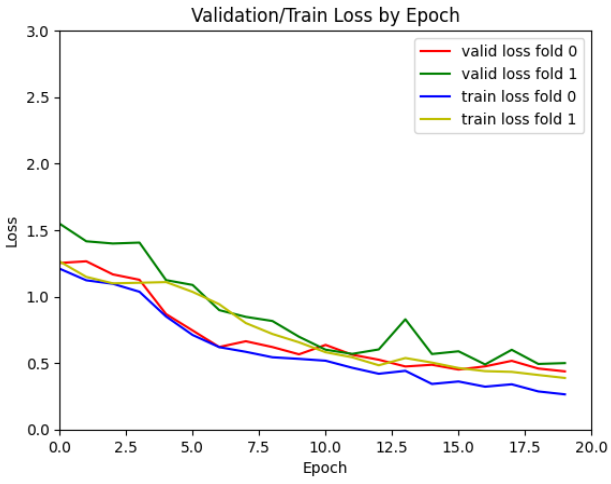
Methods of Plant Pathology Image Classification

Name(s): Derek Yang, Yung-Hsin Chao
NetID(s): mcy3, yhchao2

Summary

The objective of this project is to implement and analyze various machine learning methods employed for identifying foliar diseases in apple trees. The current method of diagnosing plant pathology is human inspections, which is costly and requires plenty of time and human resources. It takes considerable effort to train a scouting expert since the symptoms on the leaf may vary due to weather and geological conditions. In addition, misdiagnosing plant diseases can lead to further economic and environmental problems. We have experimented with 4 different models namely, MultiLayer Perceptrons, Resnet18, MobilenetV2 and Swin Transformer. We will present the results of the models mentioned above. And gave our thoughts on this subject.

A sample input image of apple leaves



The loss by epoch from resnet18

Introduction

Problem Description

The problem is provided on a kaggle competition called Plant Pathology 2020 - FGV C7 [1]. According to the competition, the specific objective of this problem is to:

1. Accurately classify a given image into different diseased categories or a healthy leaf.
2. Accurately distinguish between many diseases, sometimes more than one on a single leaf.
3. Deal with rare classes and novel symptoms
4. Address depth perception—angle, light, shade, physiological age of the leaf.

The model would take in an image as an input, and output binary classifications on 4 different categories, namely, healthy, multiple_diseases, rust, scab.

Datasets and key references

The dataset contains 3642 apple tree leaf RGB images. These images are taken under different lighting and angles. In addition, the appearance of a leaf is also affected by the age of the apple tree, temperature, and humidity. Our model will need to work accurately under numerous conditions. There will be four target labels which are healthy, multiple diseases, rust, and scab.

Below we show some example images and expected outputs.

healthy: 0 multiple_diseases: 1 rust:0 scab: 0

healthy: 1 multiple_diseases: 0 rust:0 scab: 0

A detailed discussion related to the plant pathology dataset on how they captured the images and labeled them with various diseases or abnormalities can also be found here. [2]

Hardware/Platform specifications

Most of our training primarily utilizes Google Colab. However, in circumstances where additional hardware resources are needed, we have a Google Cloud Platform provided n1-highmem-4 machine with NVIDIA V100 on board for CUDA acceleration.

Approach

The main objective of this project is to compare and contrast various models that could accurately classify the condition of the leaf. To ensure fairness and consistency, all of the models will follow the same standardized set of steps. First step, we load the Plant Pathology 2020 dataset and split it into a train set and a validation set based on the rule of two-fold cross validation. Simple data augmentation techniques, such as horizontal and vertical flips, are then implemented to expand the training set by increasing its size and introducing more diversity. This is done to enhance the model's ability to generalize and learn from a wider range of examples while enlarging the training dataset without acquiring additional data. The models were then trained on the training set using an Adam optimizer and Cross-Entropy Loss. Once the models were trained and the hyperparameters were fine-tuned, their performance was assessed on the test set. Lastly, we will compare the performance of the models, the results are shown in the ***Result*** section.

Two-Fold Cross Validation

In order to ensure a robust evaluation of our model's performance, we implemented the two-fold cross-validation technique. We divided the available plant data into two equal parts, creating distinct training and validation folds. During the first iteration, one fold was utilized as the training dataset, and the second fold acted as the validation dataset, allowing us to assess how well the model generalized to unseen examples. In the subsequent iteration, we reversed the roles of the folds. The previously used validation fold now served as the training dataset, while the former training fold became the validation dataset. This approach ensured that the model was exposed to a diverse range of plant samples and evaluated against various subsets of data. The final evaluation metrics were derived by averaging the results from both iterations, providing us with a more reliable estimate of the model's effectiveness.

Train / Test Transformation

To prepare the input images for training, a series of transformations are applied in the image preprocessing process. The initial transformation involves resizing the images to a standard size of 224 x 224 pixels using the Resize(224, 224) operation. Two transformations, namely A.HorizontalFlip(p=0.5) and A.VerticalFlip(p=0.5), are employed to randomly apply horizontal and vertical flips to the images for image augmentation.. Lastly, referenced from [4], an image normalization of mean=[0.485, 0.456, 0.406], and std=[0.229, 0.224, 0.225] is used to normalize the pixel values of the images. This step adjusts the pixel intensities according to predetermined mean and standard deviation values, ensuring that the input data is standardized before being fed into the model.

Model 1: Multilayer Perceptron with Batch Normalization and Batch drop

The first model we implemented is a traditional multilayer perceptron coupled with batch normalization and batch drop. The model includes three fully connected layers and two batch normalization layers. Drop out layers are introduced for regularization and overfitting prevention. The input layer of the model takes in images with a height and width of 224 pixels and 3 color channels, resulting in an input size of 224 * 224 * 3. The output size of the input layer is set to 256. The middle hidden layer receives 256 input features and produces 256 output features. Finally, the output layer is designed to have 256 input features and 4 output features. These 4 outputs map into the 4 classification tasks: healthy, multiple diseases, rust, and scab.

Below shows the print-out version of out implemented model:

```
(input_fc): Linear(in_features=224*224*3, out_features=256, bias=True)
(bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(dropout): Dropout(p=0.5, inplace=False)
(hidden_fc1): Linear(in_features=256, out_features=256, bias=True)
(bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(output_fc): Linear(in_features=256, out_features=4, bias=True)
```

Model 2: resnet18 conv layers with 3 fc layers

Next, we experimented with the Resnet model [3]. From the resnet18 model, we extracted the first five layers to serve as our convolutional layers. The output was then directed to three fully connected layers and two batch normalization layers. The first fully connected layer (fc1) was designed with 512 input features and 256 output features. Subsequently, the output of fc1 was passed through the first batch normalization layer (bn1) with 256 input features, which normalized the output of fc1. Continuing, the second fully connected layer (fc2) received 256 input features and produced 256 output features. The output of fc2 was then normalized by the second batch normalization layer (bn2), which had 256 input features. Finally, the output of bn2 was fed into the output_fc layer, which acted as a fully connected layer with 256 input features and 4 output features. The output_fc layer generated the final output of the model, representing the four target labels.

Note that we conducted experiments with both pretrained weights and randomized weights, interestingly, we are presented with similar performance outcomes. Based on this observation, we have made the decision to proceed with randomized weights instead of pretrained weights.

Below shows the print-out version of the connected layers after resnet18:

```
(fc1): Linear(in_features=512, out_features=256, bias=True)
(bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(fc2): Linear(in_features=256, out_features=256, bias=True)
(bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(output_fc): Linear(in_features=256, out_features=4, bias=True)
```

Model 3: mobilenet conv layers with 3 fc layers

Subsequently, we experimented with MobilenetV2 [6]. We initialized a MobileNetV2 backbone with random weights. The features are then passed through an adaptive average pooling layer to reduce the spatial dimensions to 1x1, and then flattened. This feature vector is then passed through two fully connected layers, each followed by a batch normalization layer and a ReLU activation function. The output of the last fully connected layer is passed through a final linear layer which produces the predicted probabilities for each class (in this case, there are 4 classes).

Similar to the resnet18 model, both pretrained weights and randomized weights show similar performances. We proceeded with randomized weights instead of pretrained ones.

Below shows the print-out version of the connected layers after MobilenetV2

(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc1): Linear(in_features=1280, out_features=256, bias=True)
(bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(fc2): Linear(in_features=256, out_features=256, bias=True)
(bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(output_fc): Linear(in_features=256, out_features=4, bias=True)

Model 4: Swin Transformer

We experimented with the Swin Transformer [5], a modified version of the transformer model, as our final model attempt. The Swin Transformer employs an innovative technique known as the shifted window approach to address potential complexity challenges. The Swin Transformer has shown exceptional performance on tasks such as COCO object detection and ADE20K semantic segmentation, outperforming previous models by a significant margin and has been proven to be highly versatile and serves as a reliable backbone for computer vision applications, enabling effective information modeling across different scales.

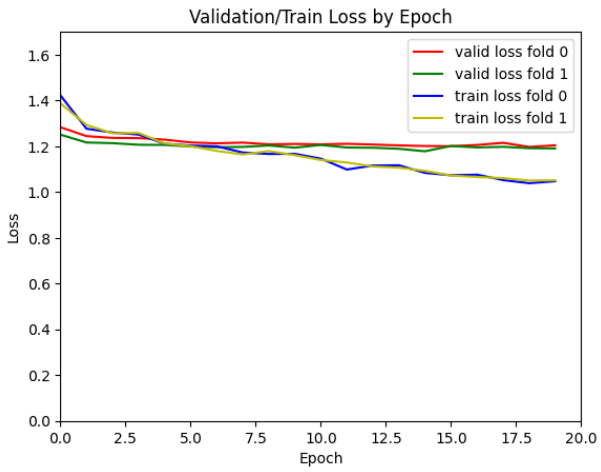
Different from the previous models, this implementation has a significant difference in performance whether it is initialized with pretrained weights or not, as a result we will include both the performance with and without pretrained weights.

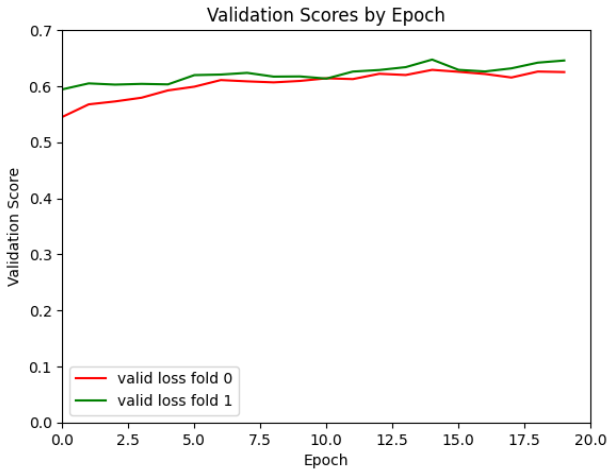
Result

Multilayer Perceptrons

Below shows the results of multilayer perceptrons losses and validation score using the Area Under the Receiver Operating Characteristic Curve (ROC AUC) over epochs.

The training time is 49 minutes and results in a score of 0.6357

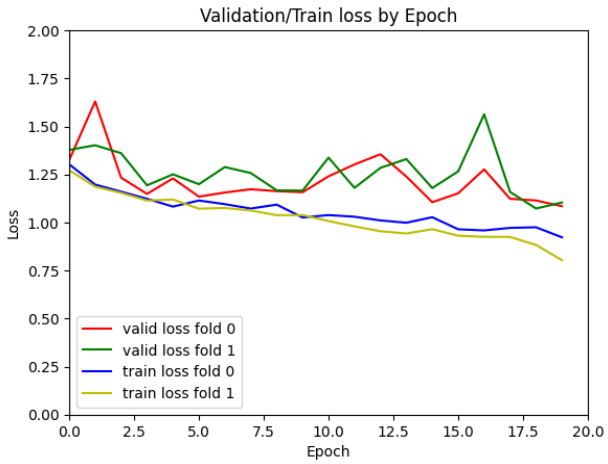


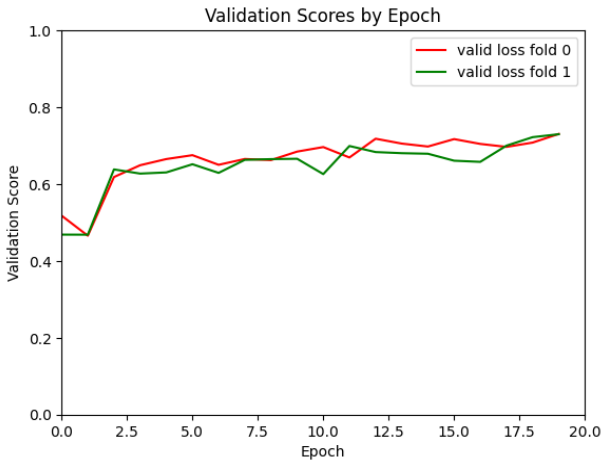


Mobilenet + FC layers

Below shows the results of our custom mobilenet with fully connected layers losses and validation score using the Area Under the Receiver Operating Characteristic Curve (ROC AUC) over epochs. Note that this model is not initialized with pretrained weights.

The training time is 50 minutes and results in a score of 0.7276

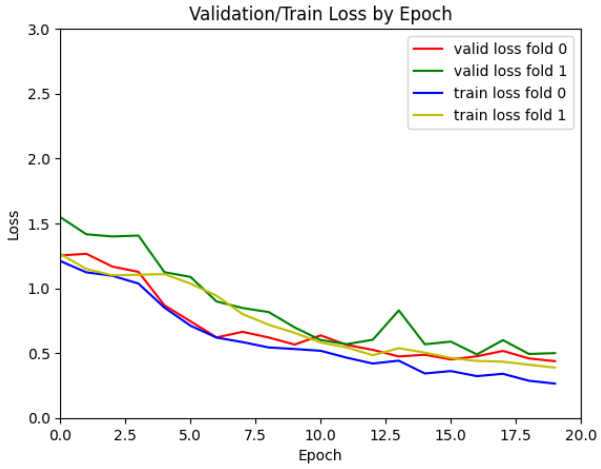


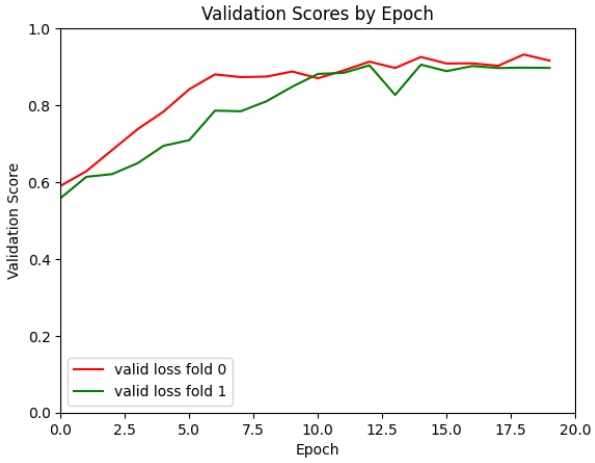


Resnet + FC layers

Below shows the results of our custom Resnet with fully connected layers losses and validation score using the Area Under the Receiver Operating Characteristic Curve (ROC AUC) over epochs. Note that this model is not initialized with pretrained weights.

The training time is 52 minutes and results in a score of 0.9017

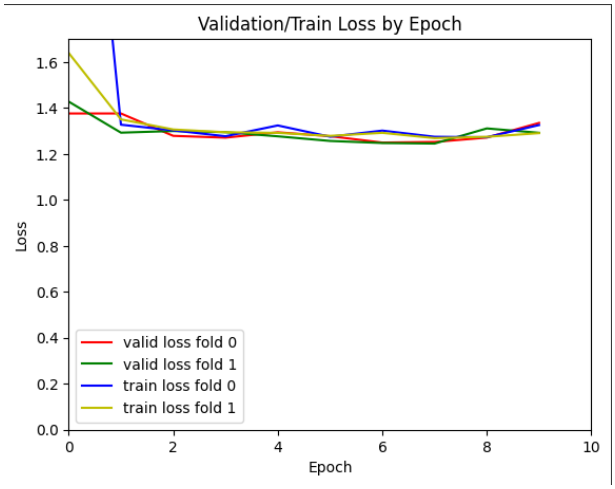


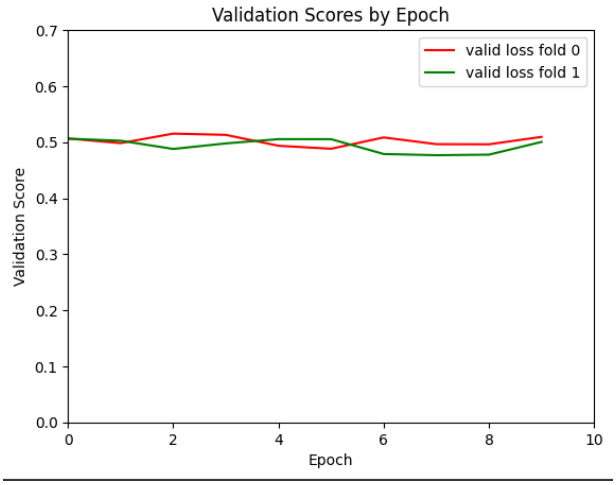


Swin Transformer

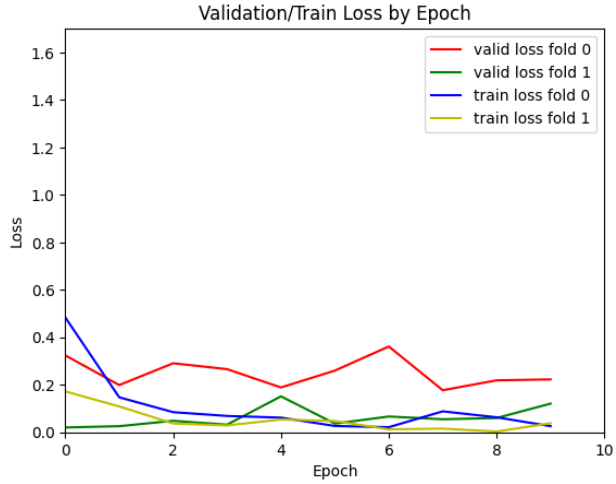
Below shows the results of the Swin Transformer losses and validation score using the Area Under the Receiver Operating Characteristic Curve (ROC AUC) over epochs.

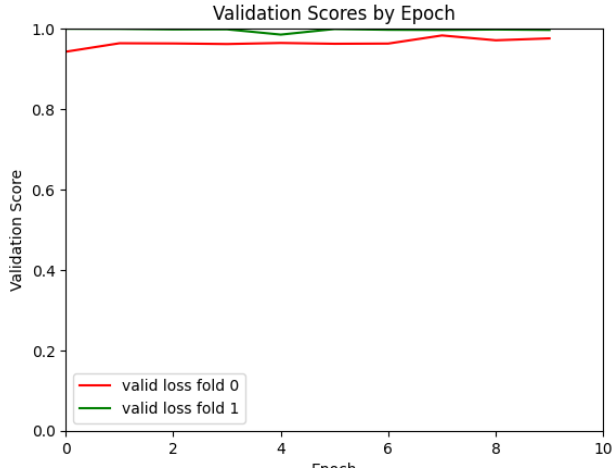
The following is the result without the pretrained weights,the training time is 56 minutes and results in an average score of 0.51.





The following is the result with the pretrained weights initialized ,the training time is 54 minutes and results in a score of 0.9794





Discussion

The performance of the MLP model we implemented seems to be stuck at a score of 0.63, signaling that it may have reached a local minimum or that there are inherent limitations with such models. When we experimented with resnet18 and mobilenet without pretrained weights, we still observed relatively good performance, with resnet18 achieving an impressive score of 0.9. Surprisingly to us, initializing with pre-trained weights on resnet18 and mobilenet didn't result in any noticeable performance improvement, leaving us uncertain about the underlying reasons. On the other hand, initializing the transformer model with pretrained weights yields a significant increase in performance from 0.51 to 0.98, showcasing the effectiveness of the transformer's attention network in image tasks coupled with transfer learning. This

further demonstrates the strong capabilities of transformer models using attention mechanisms for image analysis and image understanding.

Individual Contribution

Derek Yang: Code implementation, training and testing, documentation
Yung-Hsin Chao: Code implementation, training and testing, documentation

References

[1] https://www.kaggle.com/competitions/plant-pathology-2020-fgvc7/overview
[2] https://bsapubs.onlinelibrary.wiley.com/doi/10.1002/aps3.11390
[3] https://pytorch.org/hub/pytorch_vision_resnet/
[4] https://www.kaggle.com/code/piantic/plant-pathology-2020-pytorch-for-beginner/notebook
[5] https://github.com/microsoft/Swin-Transformer
[6] https://pytorch.org/hub/pytorch_vision_mobilenet_v2/