Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI)

AI-PROJECT-REPORT

Fruit Plant Leaf Identification and Disease Recognition

Course:- CS251

Instructor:- Nazia Shahzadi

Team member: Muhammad Adeel 2022331

Saud Khan 2022533

1. Introduction:

The primary goal of this project is to leverage deep learning techniques to build a robust model capable of accurately classifying plant diseases from images of their leaves. This endeavor is motivated by the critical need within agriculture to combat plant diseases effectively. By detecting and diagnosing diseases early, farmers and agricultural experts can implement timely interventions, thereby safeguarding crop health, enhancing yield, and minimizing agricultural losses.

Objectives:

- Develop a deep learning model capable of accurately classifying plant diseases from images of leaves.
- Enhance early detection capabilities to facilitate prompt intervention and treatment.
- Provide farmers and agricultural experts with a reliable tool for disease diagnosis and management.
- Contribute to the improvement of crop yield and agricultural productivity.
- Address the challenges associated with manual disease detection methods, such as subjectivity and inefficiency.

2. Dataset Description:

- **Content**: The dataset includes around **87,000 RGB images** featuring healthy and diseased plant leaves.
- **Classes**: Images are categorized into **38 distinct classes**, each representing different plant species affected by various diseases.
- **Augmentation**: Data augmentation techniques are employed to augment the dataset, enhancing its size and diversity.
- **Partitioning**: The dataset is split into **training and validation sets**, maintaining an 80/20 split.
- **Test Set:** A separate directory comprising **33 test images** is designated for model evaluation and deployment purposes.

3. Neural Network Architectures:

- **Custom CNN Model:** Tailored for Precision
    - **Why This Model:**
      Custom CNN architecture allows precise tuning for plant disease classification, addressing specific requirements of the task.
- **VGG16:** Leveraging pre-trained Power.
    - **Why This Model:**
    - A deep CNN Model pre-trained on ImageNet dataset.
    - Effective in capturing generic features.
    - Holds feature extractor properties.
    - Simple architecture and better performance.

- **ResNet34:** Robustness through Transfer Learning
    - **Why This Model:**
    - A deep CNN Model pre-trained on ImageNet dataset.
    - Effective in capturing intricate patterns in features in plant leaf.
    - Holds feature on shortcut connections leading to stable and faster training.

Exploring Diverse Architectures:

- **Comparative Analysis**: Experimenting with different models like VGG16 and ResNet34 allows for a comparative analysis of their performance and suitability for the plant disease classification task.

Strength:

- **Customization:** Custom CNN model offers tailored architecture catering precisely to plant disease classification requirements, enhancing precision.
- **Transfer Learning:** Utilizing pre-trained models like VGG16 and ResNet34 harnesses knowledge from large-scale datasets, improving classification accuracy through transfer learning.
- **Versatility:** Exploring multiple architectures aims to identify the most effective approach, maximizing accuracy in plant disease classification.

4. Hybrid/Ensemble Approach:

Utilizing Collective Intelligence:

- **Ensemble Approach:**
  Instead of using just one method, we teamed up different models to make our predications stronger and more reliable.
- **Combining Architectures:**
  Ensembled Custom CNN, VGG16, and ResNet34 models to exploit the unique strengths of each architecture, enhancing overall performance.

Harnessing the Power of Many:

- **Collective Intelligence:**
  By combining outputs through weighting or averaging, we harnessed the collective intelligence of diverse models, leading to more reliable and efficient classification outcomes.

Enhanced Reliability:

- **Robust Predictions:**
  Ensemble technique diminishes individual model biases and errors, resulting in more robust predictions capable of addressing various complexities in plant disease classification.

Performance Boost:

- **Improved Accuracy:**
  Leveraging the complementary strengths of multiple models contributed to a notable enhancement in classification accuracy, surpassing the capabilities of any single model alone.

5. Model Implementation:

The model implementation is documented through code snippets. Key components include data preprocessing, model architecture definition, training loop, evaluation metrics computation, and visualization functions.

Data Preprocessing

Listing the Number of unique plants in the dataset.

```
#Find the no. of unique plants and list them into a list
unique_plants = []
cl = os.listdir(data+'New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train')
for i in cl:
    x = i.split('_')
    if x[0] not in unique_plants:
        unique_plants.append(x[0])
print("Number of Unique Plants: ",len(unique_plants))
print("Unique Plants: ",unique_plants)
```

```
Number of Unique Plants:  14
Unique Plants:  ['Corn', 'Grape', 'Orange', 'Tomato', 'Pepper,', 'Apple', 'Potato', 'Cherry', 'Strawberry
```

Transformer Function for input images to resize them to 128x128 pixels and convert them to PyTorch tensors.

Loading Training and Test Dataset as Tensor

'transforms.Compose' combines these transformations into a single transformation pipeline, which can then be applied to input images or datasets. This particular pipeline first resizes the images to 128x128 pixels and then converts them to PyTorch tensors.

```
[ ] transform = transforms.Compose(
        [transforms.Resize(size = 128),
         transforms.ToTensor()])
```

```
[ ] dataset = ImageFolder(data+'New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train',transform=transform)
    test_ds = ImageFolder(data+'New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/valid',transform=transform)

    print("Number of training images: ",len(dataset))
    print("Number of testing images: ",len(test_ds))
```

```
Number of training images:  70295
Number of testing images:  17572
```

Identifying the number of unique classes.

```
[ ] #Number of classes
    num_classes = dataset.classes
    print("Number of classes: ",len(num_classes))
    print(num_classes)
```

```
Number of classes:  38
['Apple___Apple_scab', 'Apple___Black_rot', 'App
```

Setting Random seed to allow randomness to ensure reproducibility of results.

Validation Dataset and Dataloader

```
# setting a random seed allows you to
#control the randomness and ensure reproducibility of results.
random_seed = 42
torch.manual_seed(random_seed)

validation_split = 0.3
val_size = int(len(dataset) * validation_split)
train_size = len(dataset) - val_size

train_ds,val_ds = random_split(dataset,[train_size,val_size])

batch_size = 64

train_loader = DataLoader(train_ds,batch_size=batch_size,num_workers=2,shuffle=True, pin_memory=True)
val_loader = DataLoader(val_ds,batch_size=batch_size,num_workers=2,shuffle=True, pin_memory=True)
test_loader = DataLoader(test_ds,batch_size=batch_size,num_workers=2,shuffle=True, pin_memory=True)
```

Building The Model

Buiding a CNN model

Building a VGG16 model using Transfer Learning

```
class Plant_Disease_Model1(ImageClassificationBase):

  def __init__(self):
    super().__init__()
    self.network = models.vgg16(pretrained=True)
    num_ftrs = self.network.classifier[-1].in_features
    self.network.classifier[-1] = nn.Linear(num_ftrs, 38)

  def forward(self,xb):
    out = self.network(xb)
    return out
```

Building a resnet34 model using Transfer Learning

```
class Plant_Disease_Model2(ImageClassificationBase):

  def __init__(self):
    super().__init__()
    self.network = models.resnet34(pretrained=True)
    num_ftrs = self.network.fc.in_features
    self.network.fc = nn.Linear(num_ftrs, 38)

  def forward(self,xb):
    out = self.network(xb)
    return out
```

Training and Evaluation

Function to evaluate the model on validation dataset.

```python
@torch.no_grad() #decorator is used to temporarily disable
                 # gradient calculation during the evaluation.
def evaluate(model,val_loader): #used to evaluate model on val dataset
  model.eval()
  outputs = [model.validation_step(batch) for batch in val_loader]
  return model.validation_epoch_end(outputs)
  #returns the average loss and accuracy across all batches
```

Function to train the model for specified number of epochs and return training plus validation metrics

```python
# fxn to train the model for specified no. of epochs
def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
  history = []
  optimizer = opt_func(model.parameters(), lr)
  for epoch in range(epochs):
    model.train()
    train_losses = []
    for batch in tqdm(train_loader):
      loss = model.training_step(batch)
      train_losses.append(loss)
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()
    result = evaluate(model, val_loader)
    result['train_loss'] = torch.stack(train_losses).mean().item()
    model.epoch_end(epoch, result)
    history.append(result)
  return history #history listc contains training and validation metrics for each epoch
```

Training the model

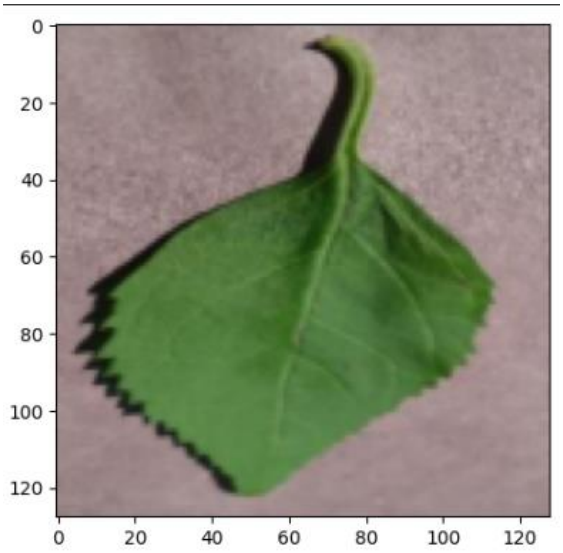
```
[ ] evaluate(model,val_loader)

    {'val_loss': 3.9106154441833496, 'val_acc': 0.024715909734368324}
```

```python
# training process for 10 epochs,learning rate: 0.001, using the Adam optimizer
history = fit(10, 0.001, model, train_loader, val_loader, opt_func = torch.optim.Adam)
```

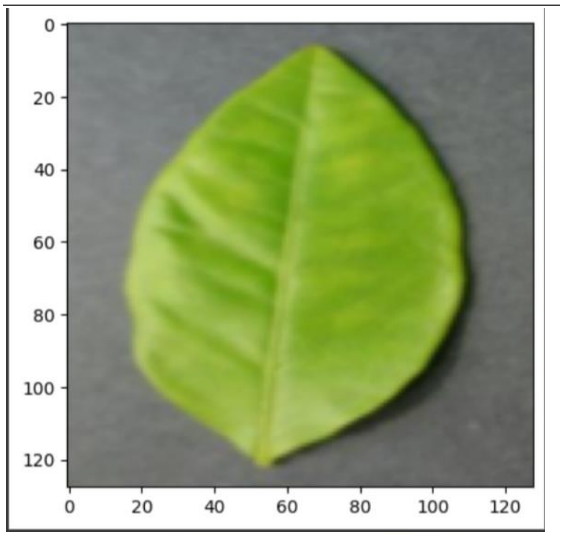
```
100% 769/769 [01:34<00:00,  7.36it/s]
Epoch [0], train_loss: 0.3628, val_loss: 0.5884, val_acc: 0.8435
100% 769/769 [01:29<00:00,  8.64it/s]
```

Prediction on some Single images from the Validation Data set



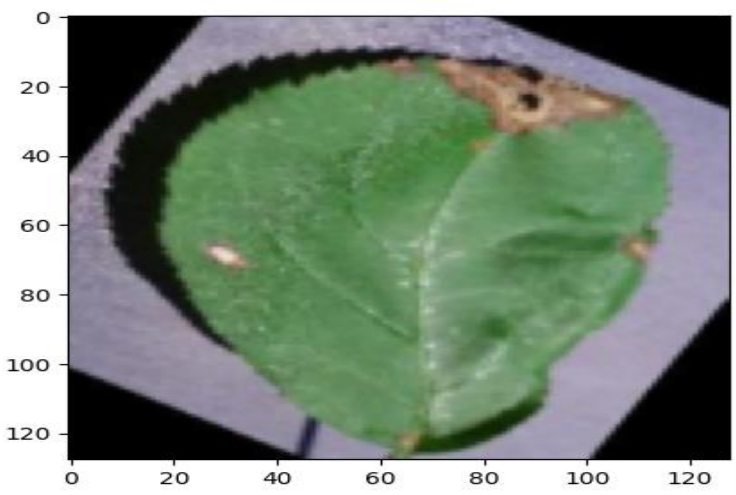
Label: Cherry_(including_sour)_healthy

Predicted: Cherry_(including_sour)_healthy



Label: Orange_Haunglongbing_(citrus_greening)

Predicted: Orange_Haunglongbing_(citrus_greening)



Label: Apple_Black_rot

Predicted: Apple_Black_rot

6- Training Procedure and Hyperparameters:

Iterative Optimization Process:

- The training process involves optimizing model performance through meticulous **hyperparameter** tuning and iterative training cycles.
- Utilizing the Adam optimizer with a **learning rate of 0.001** provides advantages such as adaptive learning rates and faster convergence.

Training Dynamics:

- **Epoch Iterations:**
  Models iterate over the dataset across multiple epochs, continuously computing loss and updating weights to minimize it, ensuring convergence towards optimal solutions.
- **Fine-Tuning Hyperparameters:**
  Key parameters like **batch size, optimizer selection, and learning rate** undergo fine-tuning for optimizing overall performance.

Adam Optimizer Advantage:

- **Efficient Training:**
  The choice of the Adam optimizer facilitates efficient and effective training by **addressing challenges like vanishing or exploding gradients**, ensuring stable and robust model updates.
- **Enhanced Accuracy and Robustness:**
  Leveraging the strengths of the Adam optimizer in deep learning applications enhances classification accuracy and robustness, critical for achieving optimal performance in plant disease classification tasks.

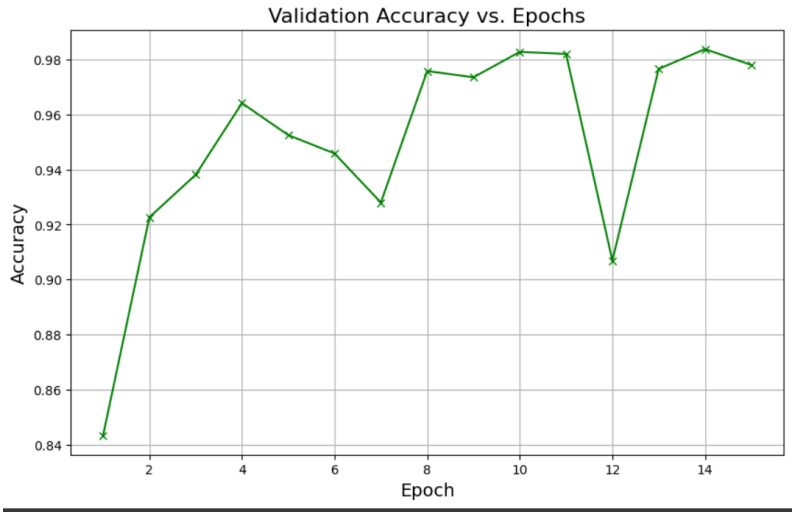
7. Evaluation Results:

The evaluation results include various metrics and visualizations:

- Accuracy:

Overall correctness of classification.

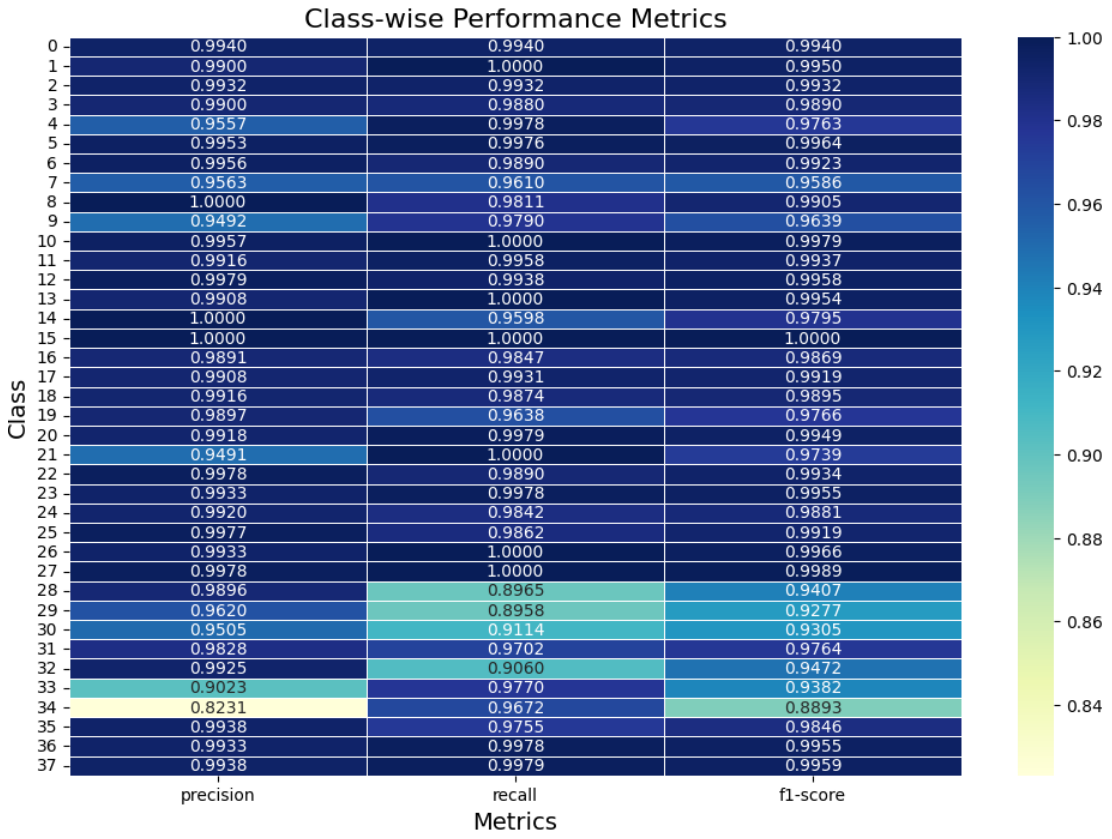
The validation accuracy increases as the number of epochs increases. This suggests that the model is learning from the training data and improving its performance on the validation set.



As the number of epochs increases, both the training loss and validation loss tend to decrease. This indicates that the model is learning to perform the task better.

- Precision, Recall, F1-score:

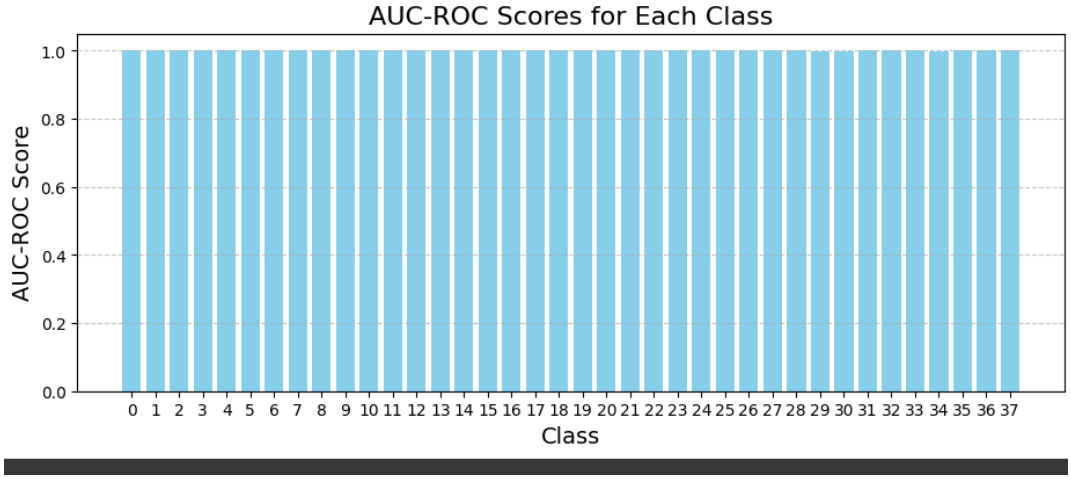
Precision measures the proportion of positive predictions that were correct.
Recall measures the proportion of actual positive cases that were correctly identified by the model.
F1-score is a harmonic mean of precision and recall, and it considers both how accurate the model's positive predictions are (precision) and how good the model is at finding all the positive cases (recall).



-AUC-ROC Score:

Measures model performance across different thresholds and class imbalances.

With every following class, we could see minimal imbalance thus proving perfect distribution among the classes.

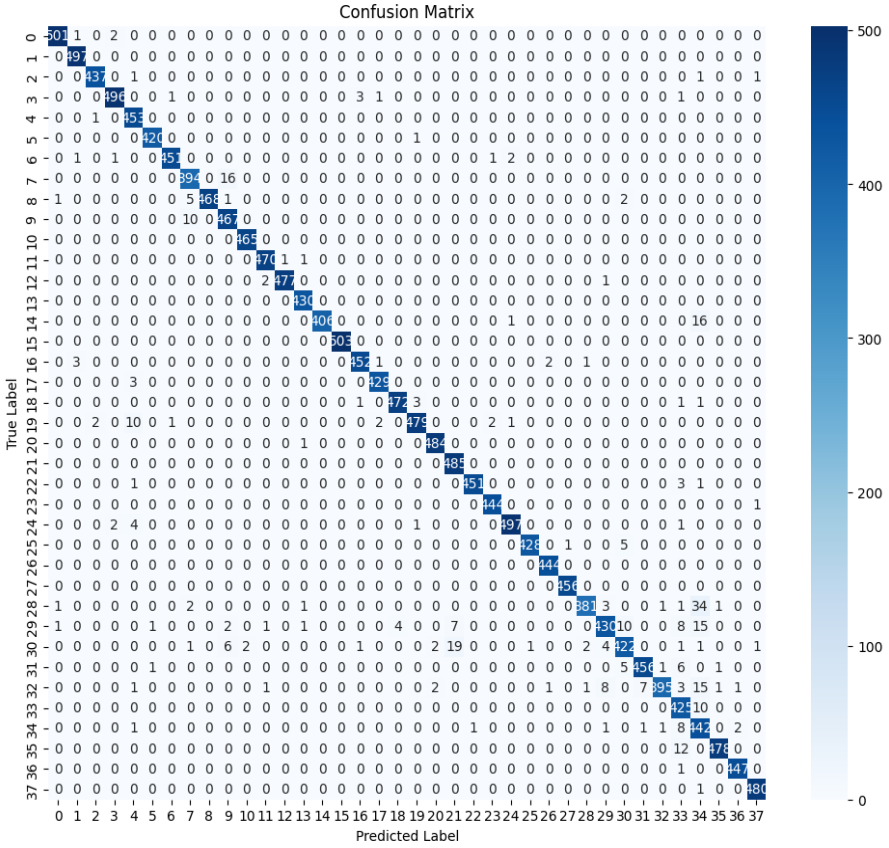


- Confusion Matrix:

A table that shows the number of correct and incorrect predictions made by a classification model.

There are many high values on the diagonal, which means the model is making a lot of correct predictions for some classes.

There are also some off-diagonal values that are high, which means the model is making some mistakes. For example, there seems to be a significant number of misclassifications between classes 14 and 15, 17 and 18, and 23 and 24.



8. Conclusion:

Our model marks a promising step forward in plant disease detection, paving the way for better crop management. Through the adept utilization of advanced deep learning methodologies and transfer learning techniques, our model demonstrates its potential to significantly augment agricultural practices by enabling timely intervention and mitigation strategies.

While our model is effective, there's still room to improve its performance and applicability in different situations. Further research and testing are needed to make it more robust, scalable, and adaptable.

As we work towards sustainable farming and food security, our model is a key tool for making agriculture stronger and more productive.