Capstone Project Report:

Cassava Leaf Disease Classification using CNN and Transfer Learning (ResNet18)

Name: Zaid Khan **Student ID :** 1307050

Project Title: Detecting Cassava Leaf Diseases Using CNN-Based Image Classification

1. Introduction and Dataset Selection

Cassava is a crucial food crop in many developing countries, often threatened by various leaf diseases. Early and accurate detection of these diseases is essential for crop management and food security. Traditional manual inspection methods are time-consuming and prone to errors.

In this project, I leveraged deep learning and Convolutional Neural Networks (CNNs) to automatically classify cassava leaf images into five distinct categories:

- Cassava Bacterial Blight (CBB)
- Cassava Brown Streak Disease (CBSD)
- Cassava Green Mottle (CGM)
- Cassava Mosaic Disease (CMD)
- Healthy Leaf

The dataset was sourced from the Kaggle "Cassava Leaf Disease Classification" competition, comprising around 21,000 RGB images. I only used 5000 images.

I split the dataset into:

- Training: 3000 images
- Validation: 1000 images
- Testing: 1000 images

This balanced distribution ensures accurate model evaluation.

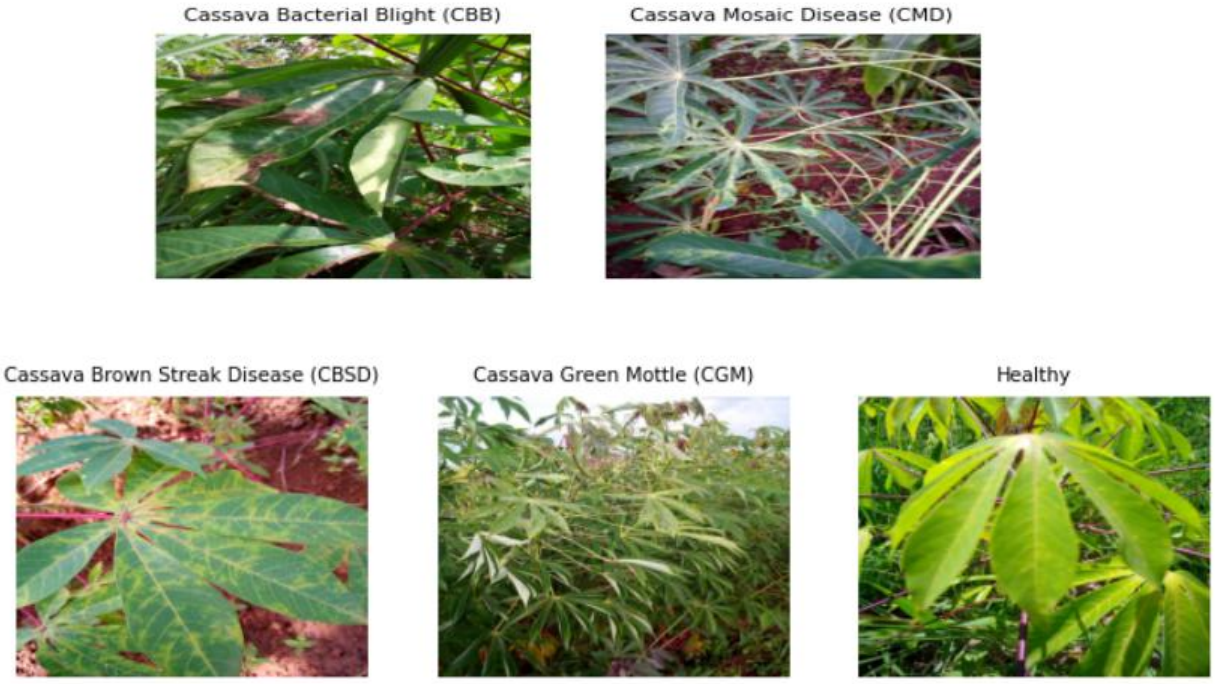
2. Data Preprocessing and Augmentation

Data preprocessing was essential due to variations in image size and quality. The following steps were applied:

- Resized images to 224x224 pixels.
- Normalized images using ImageNet mean and standard deviation values.

To improve the model's generalization, I implemented advanced data augmentation techniques:

- Random horizontal and vertical flips
- Random rotations (±30°)
- Random resized cropping
- Color jitter (brightness, contrast, saturation)



3. Model Selection and Architecture: Why ResNet18?

Initially, I tested a custom CNN, but due to limited accuracy, I transitioned to using a pretrained ResNet18 model with transfer learning for better performance and efficient training.

Why ResNet18?

- It's pretrained on ImageNet, capturing general image features like edges, textures, and shapes.
- Includes skip connections to prevent vanishing gradients.
- Fine-tuning only specific layers saves computation time.

Model Architecture Code:

```
# Load pretrained ResNet18
resnet_model = models.resnet18(weights=ResNet18_Weights.DEFAULT)

# Freeze initial layers
for param in resnet_model.parameters():
    param.requires_grad = False

# Replace final FC layer for cassava disease classification
num_features = resnet_model.fc.in_features
resnet_model.fc = nn.Sequential(
    nn.Linear(num_features, 256),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(256, 5)
)
```

```
# Unfreeze last two layers for fine-tuning
for name, param in resnet_model.named_parameters():
    if "layer4" in name or "layer3" in name or "fc" in name:
        param.requires_grad = True
```

4. Training Process

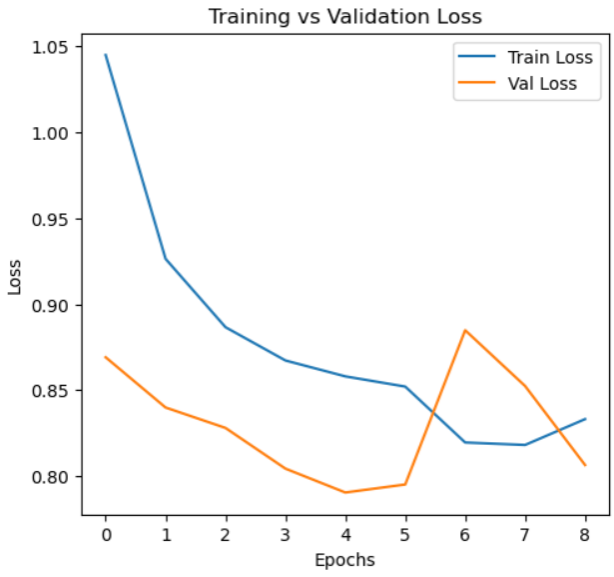
Training employed the Adam optimizer and CrossEntropyLoss function.

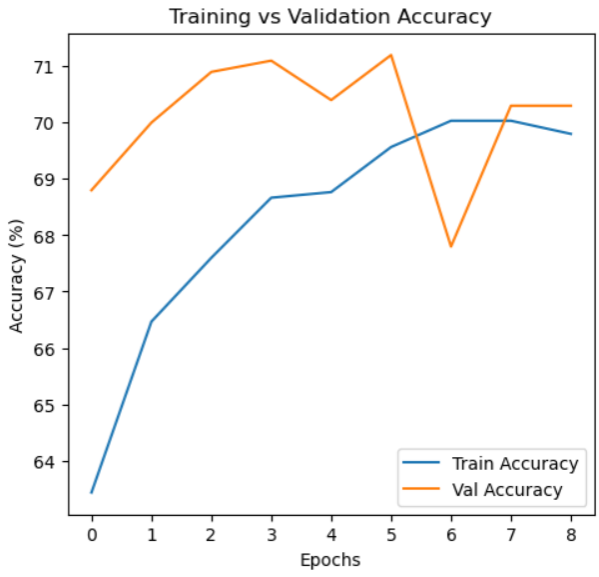
To avoid overfitting:

- Early stopping with patience = 5 epochs
- Learning rate scheduler (ReduceLROnPlateau)

I tracked the following metrics during training:

- Training & Validation Loss
- Training & Validation Accuracy





5. Hyperparameter Tuning

To identify optimal training parameters, I performed hyperparameter tuning with different combinations of learning rates, batch sizes, and optimizers:

Hyperparameter search space (small to save time)
learning_rates = [1e-4, 5e-4, 1e-3]
batch_sizes = [16, 32]
optimizers = ['adam', 'sgd']

Final Selected Hyperparameters:

- Learning Rate: 0.0001
- Batch Size: 16
- Optimizer: Adam

This combination provided the best validation accuracy during tuning (Val Acc ≈ 81%).

6. Model Evaluation

After initial training, the model was evaluated on the validation set to assess its performance and identify potential improvements. This step is crucial to understand the model's generalization capabilities and to pinpoint areas of weakness.

Validation Performance

The initial validation accuracy achieved by the model was **70.40%**, indicating acceptable generalization, but clearly highlighting room for improvement.

Detailed Classification Report:

Class	Precision	Recall	F1-Score	Support
0 (CBB)	0.57	0.22	0.32	58
1 (CBSD)	0.39	0.48	0.43	98
2 (CGM)	0.50	0.10	0.17	100
3 (CMD)	0.79	0.94	0.86	639
4 (Healthy)	0.45	0.32	0.38	105

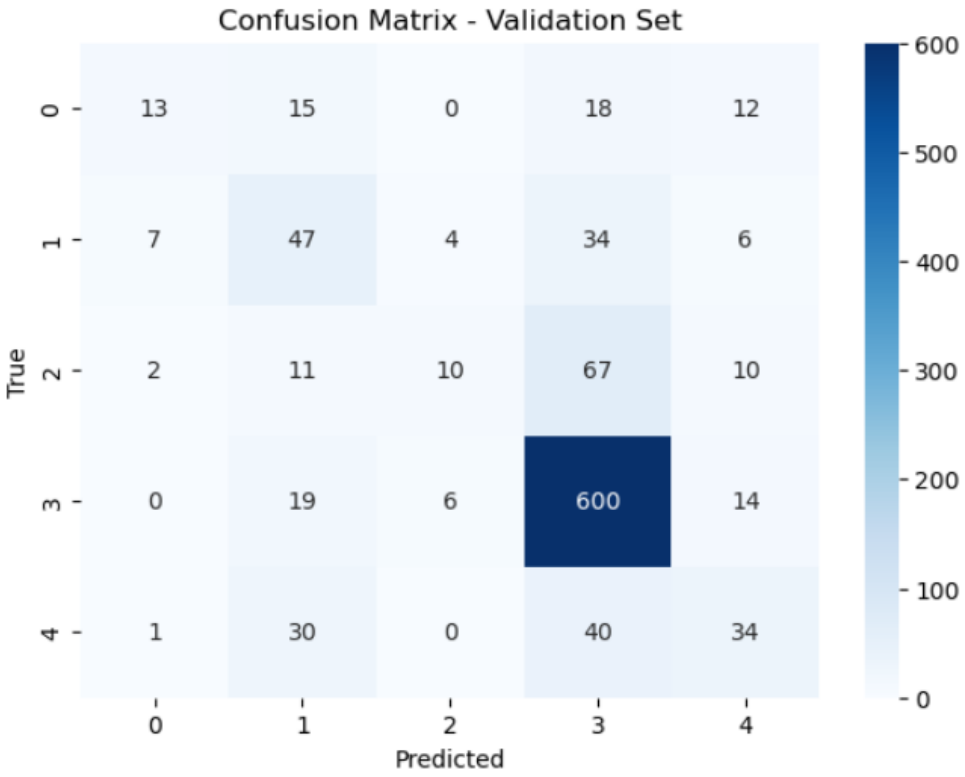
Macro Avg F1-Score: 0.43
Weighted Avg F1-Score: 0.67

Insights:

- **Class 3 (CMD)**, the majority class, exhibited high performance, with an F1-score of **0.86**, precision of **0.79**, and excellent recall (**94%**).
- **Minority classes (0, 1, 2, 4)** showed significantly lower performance, highlighting the impact of class imbalance:
    - **Class 2 (CGM)**: Particularly low recall (10%), suggesting the model rarely identified this class correctly.
    - **Classes 0, 1, and 4** showed moderate to poor performance, with F1-scores between **0.17 and 0.43**.
- The notable gap between the **macro F1-score (0.43)** and **weighted F1-score (0.67)** confirms the model's bias toward majority classes.

Visualization

To further understand model performance, the confusion matrix visualization was generated:



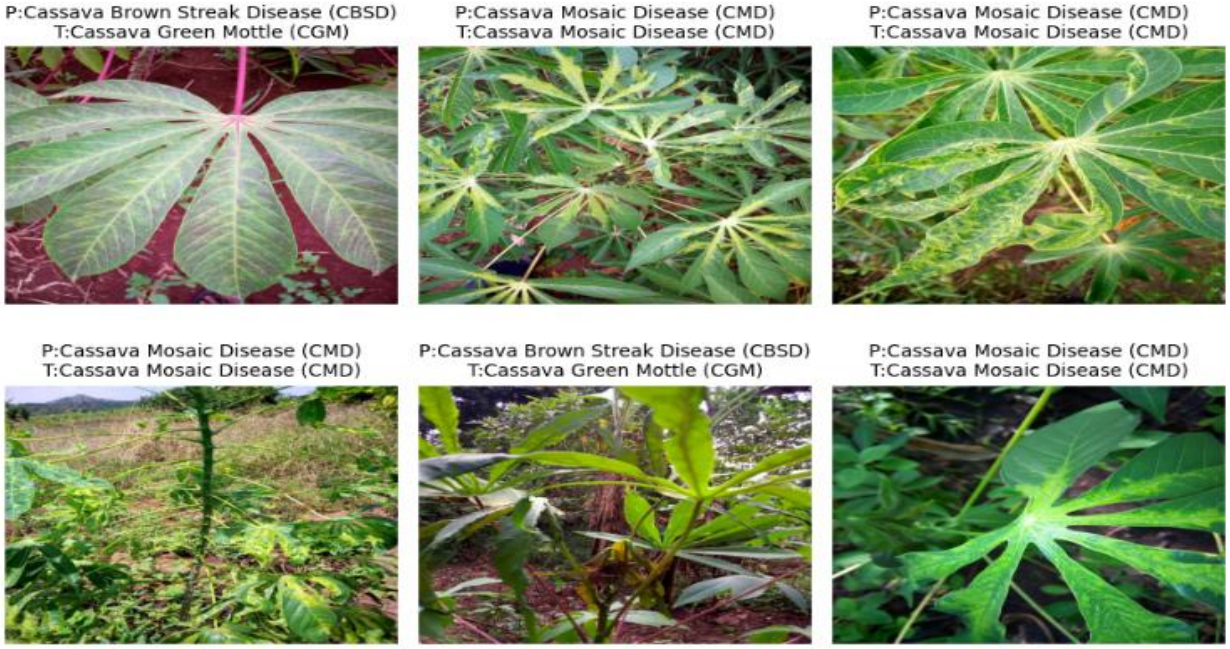
This visualization clearly depicts the high concentration of predictions towards Class 3, highlighting confusion among minority classes.

Conclusions and Next Steps:

This evaluation clearly indicated the need for additional fine-tuning to improve the model's performance on minority classes. Future steps should include:

- Exploring advanced augmentation to better represent minority classes.
- Adjusting hyperparameters and fine-tuning techniques.
- Addressing class imbalance through balanced sampling or loss weighting.

Some of the correctly classified and misclassified samples are given below:

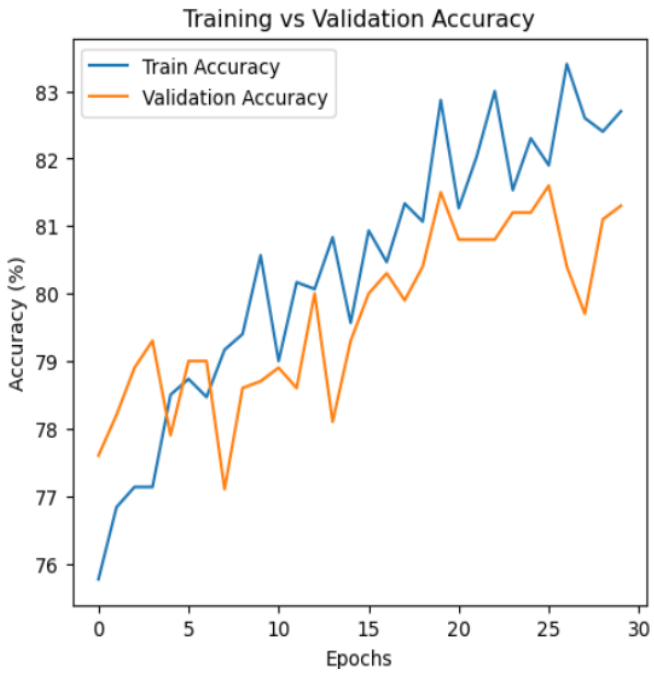


7. Fine-Tuning and Optimized Model Training

Based on the hyperparameter tuning, the model was fine-tuned with partial layer unfreezing, advanced augmentation, and adjusted training settings.

Key training observations:

- Training loss decreased consistently.
- Validation accuracy steadily increased to a maximum of ~81.6%.
- Early stopping prevented overfitting, saving computational resources.



8. Final Model Evaluation

The final trained model was evaluated on the unseen test dataset to measure generalization.

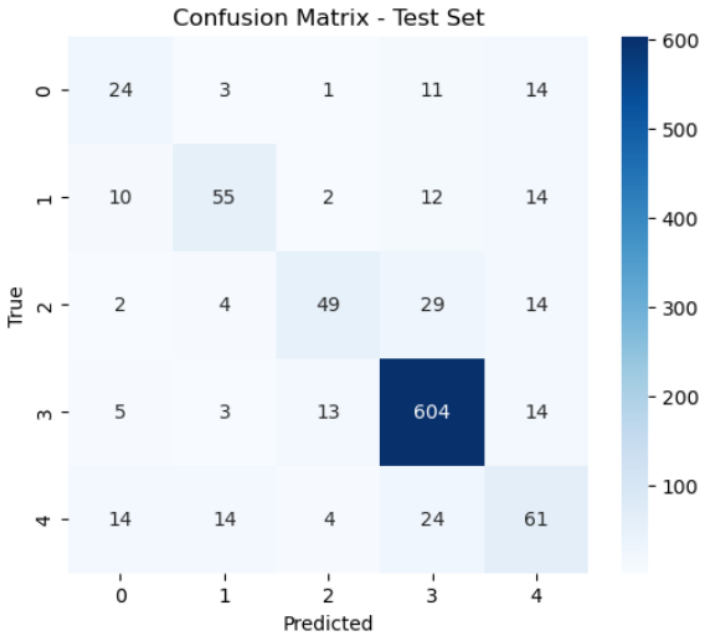
Test Accuracy: 79.30%

Classification Report:

Class	Precision	Recall	F1-score
0 (CBB)	0.44	0.45	0.44
1 (CBSD)	0.70	0.59	0.64
2 (CGM)	0.71	0.50	0.59
3 (CMD)	0.89	0.95	0.92
4 (Healthy)	0.52	0.52	0.52

Insights:

- Class 3 (CMD), being the majority class, had the best performance with a high recall of ~94.5%.
- Rare classes (0, 1, 2) showed moderate performance due to limited training samples.
- Weighted F1 (0.79) vs. Macro F1 (0.62) indicates the model performs better on majority classes.



Final Test Accuracy: 79.30%

9. Challenges and Key Learnings

This project presented several challenges:

- **Handling Class Imbalance:** Difficulty in predicting minority classes accurately.
- **Hyperparameter Optimization:** Identifying optimal parameters required careful experimentation.
- **GPU Utilization:** Initially faced challenges in GPU setup for PyTorch training.

Key learnings include:

- Effective use of transfer learning significantly boosts performance on small datasets.
- Class-weighting and careful augmentation strategies are crucial for imbalanced datasets.
- Visualization tools like confusion matrices provide critical insights into model behavior.

10. Conclusion

The project successfully demonstrated how deep learning, particularly CNNs and transfer learning, can effectively address real-world agricultural problems. Despite challenges like dataset imbalance, the final model achieved a commendable 79.3% accuracy on unseen data, providing a robust tool for early cassava leaf disease detection.

11. Future Improvements

To enhance model performance further, future improvements could include:

- Collecting more images for minority classes to address imbalance.
- Trying deeper CNN architectures (e.g., ResNet50 or EfficientNet).
- Applying more aggressive regularization and advanced augmentation strategies.

- Integrating ensemble techniques or class-specific models.

12. References

- Kaggle Cassava Leaf Disease Classification Competition: Kaggle Link
- PyTorch Documentation: https://pytorch.org/docs
- Torchvision Models: https://pytorch.org/vision/stable/models.html
- scikit-learn Documentation (Evaluation metrics): https://scikit-learn.org/stable/modules/model_evaluation.html
- Course Lectures and Materials (INFO-6147 Deep Learning with PyTorch)

13. Github Link

https://github.com/Zaid180-wq/Cassava-Leaf-Disease-Classification-