1. Introduction

In this project we develop and evaluate a convolutional neural network—**ColabOptimizedCNN**—to classify plant leaf images into multiple disease categories. We implemented data loading, model training with early stopping, advanced evaluation (precision-recall, ROC, confusion matrix), feature-space visualization (PCA, t-SNE, UMAP), and model complexity analysis. The following document breaks down each component in simple, detailed steps, then interprets the results shown in the attached charts.

2. Environment Setup and Data Preparation

1. **Package installation**: Torch, torchvision, Matplotlib, scikit-learn, UMAP, etc.
2. **Mount Google Drive**: so we can load and save large datasets and models directly.
3. **Zip extraction**: we locate and unzip `Plant_leaf_diseases_dataset_with_augmentation.zip` into `/content/extracted_dataset`.
4. **Automatic root detection**: `find_dataset_root(...)` scans subfolders until it finds class directories containing images.
5. **Dataset exploration**: prints the first 10 classes and their image counts, plus the total number of images and classes.
6. **Train/Validation/Test split**: 70% train, 15% validation, 15% test, with a fixed random seed for reproducibility.

3. Image Transforms and DataLoaders

- **Dynamic sizing**: If a GPU is available, images are resized to 128×128 then center-cropped to 112×112; on CPU we use 96→84 to speed up training.
- **Augmentations** (training only): random horizontal flips, rotations, brightness/contrast jitter to improve generalization.
- **Normalization**: standard ImageNet means and standard deviations.
- **DataLoaders**: batch size 64 on GPU (32 on CPU), two workers, pinned memory if possible, and `SubsetRandomSampler` for each split.

4. Model Architecture: ColabOptimizedCNN

A lightweight, mobile-friendly CNN inspired by depthwise separable convolutions:

Stage	Layers	Purpose
1	Conv2d(3→32), BatchNorm, ReLU6	Initial feature extraction
2	_make_depthwise_block(32→64, stride=1)	Depthwise + pointwise conv block
3	_make_depthwise_block(64→128, stride=2)	Downsampling block
...	Additional depthwise blocks up to 512 channels	Progressive deepening
Final	AdaptiveAvgPool → Flatten → Dropout → Linear(512→256) → ReLU → Dropout → Linear(256→num_classes)	Classification head

- **Depthwise separable blocks** split spatial and channel mixing, reducing parameters.
- **Initialization**: Kaiming for conv layers, constant for batch norm, small normal for linear layers.
- **Feature hook**: `get_features()` returns the 512-dim vector before the classifier for later clustering.

Total parameters: ~410 k (≈1.57 MB) **Trainable parameters**: 410 k

5. Training Procedure

- **Optimizer**: AdamW with weight decay 1e-4, initial LR 0.001.
- **Scheduler**: StepLR reduces LR by ½ every 7 epochs.
- **Early stopping**: stops if validation accuracy does not improve for 5 consecutive epochs.
- **Logging**: per-batch loss every 50 batches, plus per-epoch summary:
    - Train loss & accuracy
    - Val loss & accuracy
    - Learning rate
    - Early stopping counter

Training run: 20 epochs, best validation accuracy ≈ 0.9882 at epoch 20.

6. Test-set Evaluation Metrics

After loading the best model checkpoint, we compute:

- **Loss** on test set: ~0.0364
- **Accuracy**: 0.9873
- **Macro Precision**: 0.9849
- **Macro Recall**: 0.9839
- **Macro F1-score**: 0.9842
- **Micro Precision**: 0.9873
- **Micro Recall**: 0.9873
- **Micro F1-score**: 0.9873

These high scores reflect excellent overall and per-class performance.

7. Precision–Recall and ROC Curves

Precision–Recall (first image set)

- **Micro-averaged**: nearly perfect curve hugging the top-right corner (precision ≈ 1.0 for recall up to 0.99).
- **Macro-averaged**: similarly high, indicating all classes perform well.
- **Per-class (top 5)**: each of the first five classes shows near-unit precision and recall.

ROC (second image set)

- **Micro AUC**: 1.000
- **Macro AUC**: ≈ 0.9999
- **Per-class AUC** (top 5): all AUC = 1.000.

Interpretation: The model separates positive vs. negative examples perfectly across classes, with negligible overlap.

8. Feature-Space Clustering (third image set)

We extract 512-dim feature vectors for 1,000 test samples, then visualize with:

1. **PCA**: first two components capture ~28.1% variance. Clusters start to form but many overlap.
2. **t-SNE**: tighter, well-separated clusters showing that learned features discriminate classes.
3. **UMAP**: similar separation, sometimes revealing substructure within classes.
4. **Class distribution** bar chart: shows sample counts per class in this subset (some classes more frequent than others).

Takeaway: feature extractor produces linearly separable clusters for most classes, confirming strong representation learning.

9. Model Complexity Analysis

From the printed layer-by-layer breakdown:

- **Conv2d layers**: ~64.5% of total parameters
- **BatchNorm2d layers**: ~1.1%
- **Linear layers**: ~34.4%

Other plots:

- **Final layers bar chart**: shows that the two final linear transforms (256→num_classes) contain the majority of parameters.
- **Memory usage**: parameters and gradients each ≈1.56 MB, activations estimate ≈2.57 MB per batch.
- **Trainable vs. non-trainable**: all parameters are trainable.

Conclusion: At ~410 k parameters and <2 MB size, this model is compact and suitable for resource-limited environments.

10. Training Dynamics and Overfitting Check (last image set)

1. **Loss curves**: training and validation loss both decrease smoothly, with validation always below training after epoch 2, indicating no overfitting.
2. **Validation accuracy**: climbs from ~0.73 to ~0.99, plateauing near the best value.
3. **Learning rate schedule**: starts at 1e-3, halves at epoch 7 to 5e-4, then halves again at epoch 14 to 2.5e-4.
4. **Overfitting indicator** (val − train loss): stays negative or near zero, confirming good generalization.
5. **Test-set metrics bar chart**: re-plots accuracy, precision, recall, F1 for macro and micro, all near 0.99.
6. **Convergence analysis**: moving average of val-accuracy shows stable convergence by epoch 10.
7. **Training summary box**: lists final epochs, best val acc, test acc, total params, size, final LR, and approximate training time (~40 min).

11. Sample Predictions

The final section (not plotted as a chart) prints 5 random test images with:

- **True vs. predicted class**
- **Confidence score** of top prediction
- **Top 3 predicted classes** with probabilities

All sample predictions show correct labels with high confidence (≈1.000), further illustrating the model's reliability.

---

12. Conclusion

- **Performance**: 98.7% test accuracy with balanced macro/micro metrics.
- **Efficiency**: <1 MB model, fast inference, suitable for deployment on mobile/edge.
- **Interpretability**: clear feature clustering, simple architecture, and transparent complexity breakdown.

Future work could explore real-time deployment on smartphones, continued data augmentation for robustness, or lighter backbones for even smaller footprints.

Prepared by Group 13 – BSc Computer Science, June 2025