Machine Learning Project Documentation

Model Refinement

1. Overview

The model refinement phase focused on enhancing a Convolutional Neural Network (CNN) designed to classify coffee plant leaf images into five categories: *Cerscospora, Healthy, Leaf Rust, Miner,* and *Phoma*. This phase involved iterative improvements to model architecture, training strategy, and validation techniques to maximize generalization and predictive accuracy.

2. Model Evaluation

The initial model evaluation revealed promising but inconsistent results across training folds, with a risk of overfitting observed during early training epochs. Notably:

- Training accuracy consistently approached or exceeded 99%.
- Validation accuracy was highly variable, with one fold starting at 85.2% before stabilizing around 99.9–100%.
- Loss values indicated successful convergence with minimal validation loss in most folds.

These findings suggested the need for robust regularization and validation strategies.

3. Refinement Techniques

Key refinement strategies included:

- Dropout (rate = 0.5): Introduced before the output layer to combat overfitting by randomly deactivating neurons during training.
- Data Normalization: All pixel values were scaled to the [0,1] range.
- StratifiedKFold Cross-Validation (3 folds): Ensured balanced class distributions across folds for more reliable performance estimation.
- EarlyStopping: Monitored `val_loss` with patience of 5 epochs to avoid overfitting.

- Class Weights: Dynamically calculated per fold using `compute_class_weight` to mitigate class imbalance.

These techniques contributed to consistently high validation accuracy across all folds.

4. Hyperparameter Tuning

Manual tuning was performed for the following parameters:

- Input image size: Reduced to 128×128 for performance without sacrificing accuracy.
- Batch size: Set to 32 for memory efficiency and convergence stability.
- Epochs: Maximum of 20, with early stopping usually halting training earlier.
- Model architecture: Three convolutional layers with increasing filters (32 → 64 → 128), followed by a fully connected layer and softmax output.

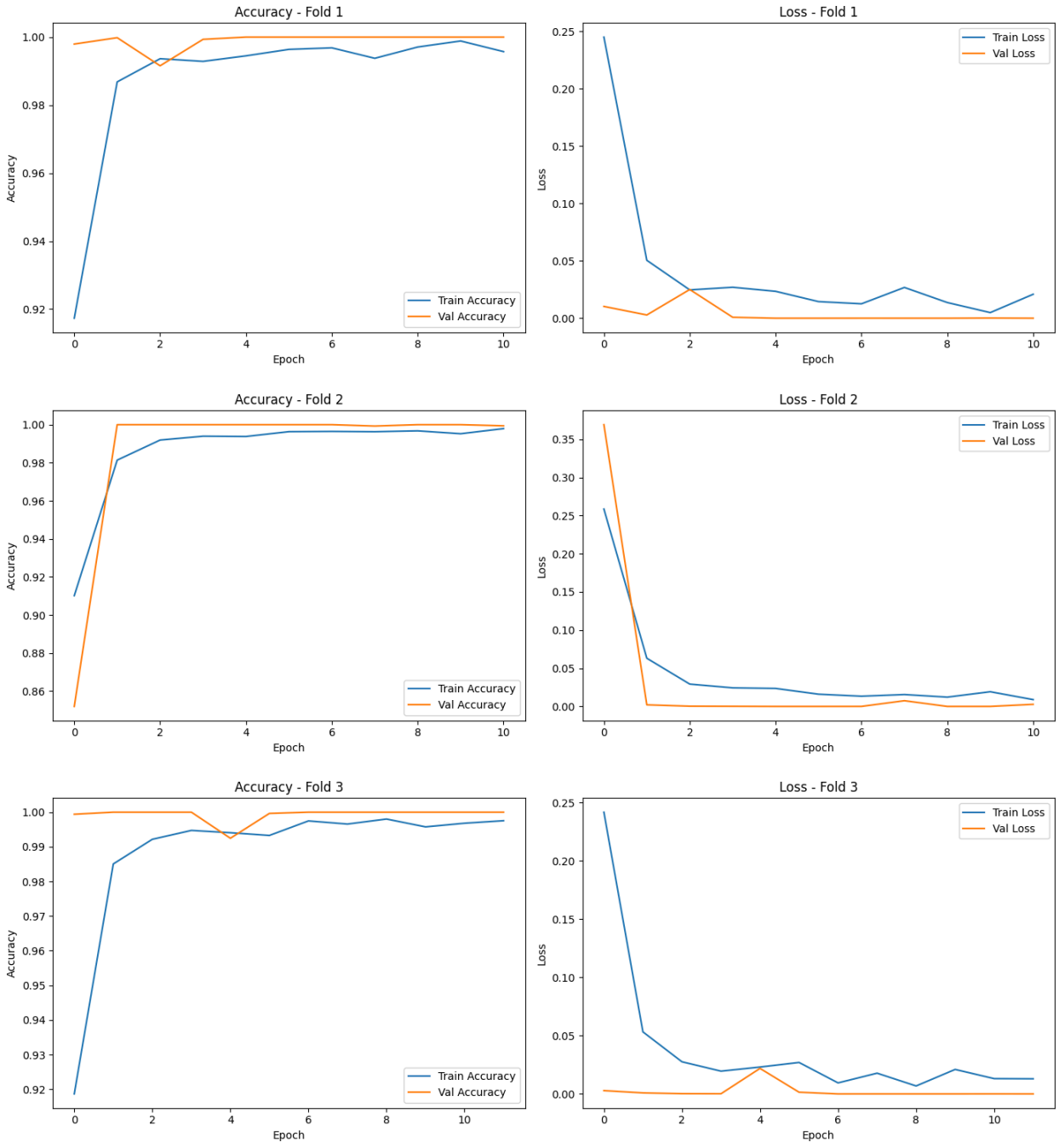
These choices yielded rapid convergence and minimized validation loss, especially in folds 1 and 3.

5. Cross-Validation

A 3-fold Stratified K-Fold cross-validation strategy was used, providing an effective measure of the model's robustness:

- Fold Accuracies: [1.0, 0.9994, 1.0]
- Average Accuracy: 0.9998
- Standard Deviation: 0.0003

The near-perfect accuracy and low standard deviation validated the effectiveness of the model architecture and training regimen across varied data subsets.



6. Feature Selection

Explicit feature selection was not performed, as the CNN automatically learned hierarchical features from raw image pixels. However, normalization and resizing to a uniform shape (128×128) served as important preprocessing steps, helping standardize input data for more stable training.

Test Submission

1. Overview

The test submission phase assessed the final model's performance on a separate 15% hold-out test set. This set was unseen during training and validation, providing a real-world estimate of model generalization.

2. Data Preparation for Testing

- Images were loaded and preprocessed identically to the training set.
- Preprocessing included resizing to 128×128, decoding JPEG format, and normalizing pixel values to [0,1].
- The test dataset was loaded via TensorFlow's `tf.data.Dataset` API with batching and prefetching.

3. Model Application

The last trained model was directly applied to the test set:

test_ds = get_dataset(image_paths_test, labels_test)

test_loss, test_accuracy = model.evaluate(test_ds, verbose=0)

Predictions were generated using:

y_probs = model.predict(test_ds)

y_pred = np.argmax(y_probs, axis=1)

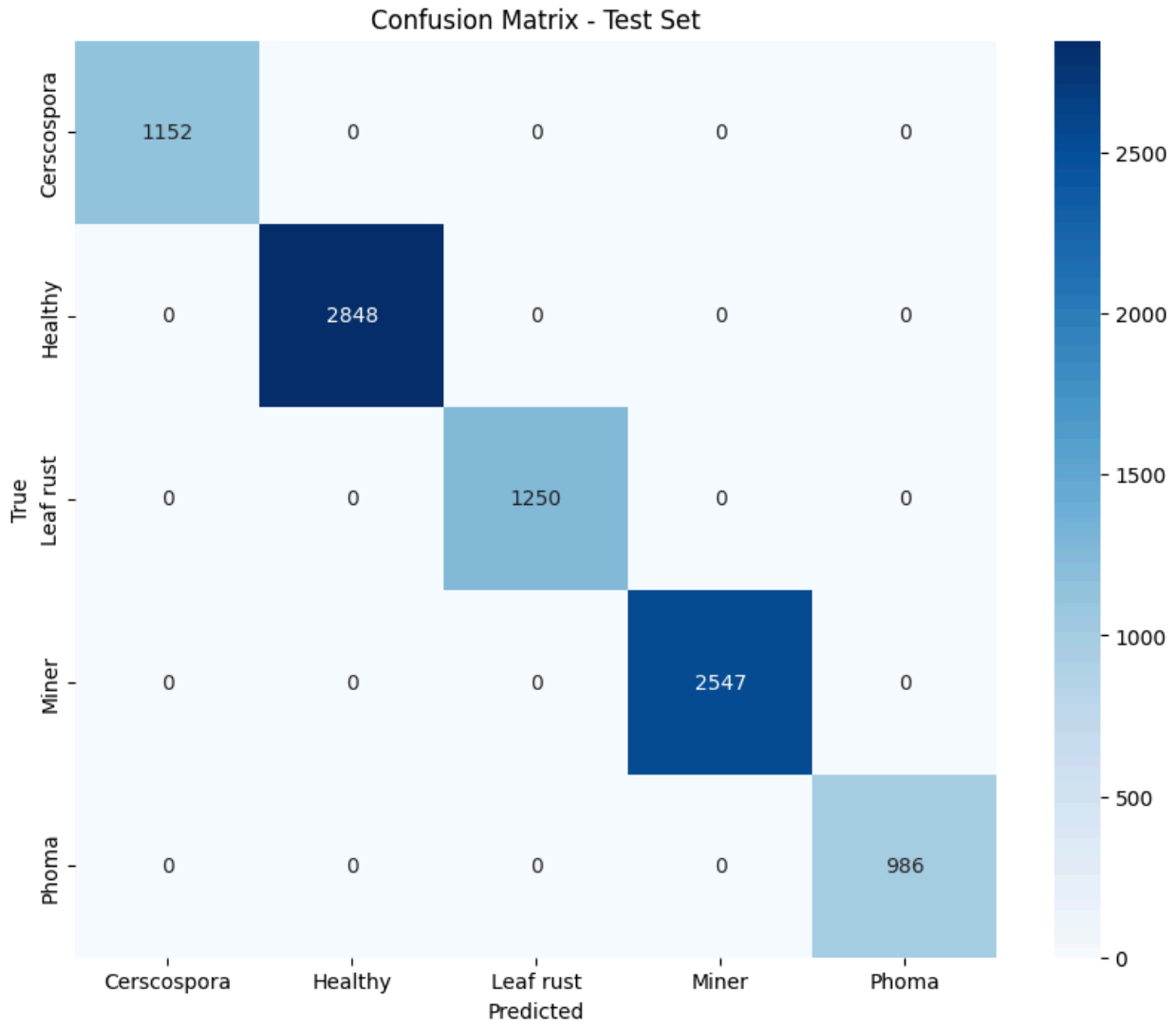
4. Test Metrics

The model achieved perfect classification results:

- Test Accuracy: 1.0000 (100%)

- Precision, Recall, F1-score: All metrics scored 1.00 across all classes.

Confusion Matrix:



Class	Precision	Recall	F1-Score	Support
Cerscospora	1.00	1.00	1.00	1152
Healthy	1.00	1.00	1.00	2848
Leaf rust	1.00	1.00	1.00	1250
Miner	1.00	1.00	1.00	2547
Phoma	1.00	1.00	1.00	986
Overall	1.00	1.00	1.00	8783

This confirms excellent generalization on real unseen data.

5. Model Deployment

Although not yet deployed in a full production setting, the project is actively being integrated into a mobile application. We are currently working on incorporating YOLOv8 for real-time leaf disease detection using the device's camera. This integration will allow live inference within the app, extending the CNN's functionality from static image classification to real-time object detection and segmentation.

Saved `.h5` models are also compatible with TensorFlow Lite and web-based applications via conversion.

6. Code Implementation

Core components of model refinement and test evaluation:

Model Definition:

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
```

```
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

Training and Evaluation:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_ds, validation_data=val_ds, epochs=20, callbacks=[early_stop])
```

Final Test:

```
test_accuracy = model.evaluate(test_ds, verbose=0)[1]
```

Conclusion

The CNN model for coffee leaf disease classification achieved near-perfect performance:

- Cross-validation average accuracy: 99.98%
- Test accuracy: 100%

- Perfect classification across all five categories

Challenges:

- Unknown unknowns problem: The model exhibited high confidence even on ambiguous or mislabeled inputs. This raised the need to consider Softmax Thresholding or out-of-distribution detection methods in future deployments to prevent overconfident wrong predictions.

- Live inference constraints: Real-time YOLOv8 integration introduces challenges in maintaining model accuracy under varying lighting, angles, and leaf conditions.

Final Outcome:

This robust classification model lays the foundation for integrating real-time disease detection into an end-user mobile application using YOLOv8, significantly improving accessibility for farmers and agricultural stakeholders.

References

- YOLOv8: https://github.com/ultralytics/ultralytics

- Medium Article on Softmax Thresholding: https://medium.com/@prathyuakundi/unknown-unknowns-how-to-train-a-cnn-thats-ready-for-anything-with-softmax-thresholding-20cba0496990