Machine Learning Project Documentation
Deployment

1. Overview

In the deployment phase, the trained Keras model was first serialized and then converted into TensorFlow Lite (`.tflite`) format to optimize it for mobile deployment. The converted TFLite model was then integrated into a Flutter-based mobile application, allowing real-time, on-device predictions for plant disease detection. The deployment did not rely on any external cloud infrastructure, ensuring low-latency and offline functionality.

2. Model Serialization

The model was first trained and saved in Keras format (`.h5`). After confirming its accuracy, the model was converted to TFLite format (`.tflite`) using TensorFlow's conversion tools. This format reduces model size and inference latency, making it suitable for mobile devices. The converted model (`plant_disease_model.tflite`) was placed in the Flutter app's `assets/models/` directory for direct use with the `tflite_flutter` plugin.

3. Model Serving

The serialized TFLite model is loaded and run directly on the mobile device using the `tflite_flutter` Dart package. Here's how serving is implemented:

- The `ModelProvider` class handles loading and initializing the model using:

```
_interpreter = await Interpreter.fromAsset(
  'assets/models/plant_disease_model.tflite',
  options: InterpreterOptions()..threads = 4,
);
```

- The model runs inference on preprocessed image data via:

```
_interpreter!.run(reshapedBuffer, outputBuffer);
```

This local, on-device serving approach removes the need for a backend server, offering faster performance and improved privacy.

4. API Integration

There was no external API created for this deployment. All model interactions happen on-device, so no API keys or HTTP endpoints are involved. Input to the model is an image file, and the output is a structured object (`PredictionResult`) containing the disease type, confidence score, suggestions, and metadata.

5. Security Considerations

Since the model is served entirely on-device and does not rely on any external services or APIs, no security measures such as authentication, authorization, or encryption were implemented or required. No user data is transmitted or stored externally, minimizing security risks.

6. Monitoring and Logging

While no external monitoring tools are used, a basic logging and tracking system is implemented within the app using a `HistoryProvider`. Each prediction result, including timestamp, accuracy, and disease type, is stored locally:

```
historyProvider.addResult(result);
```

This approach allows users to review their prediction history. Although it doesn't include advanced monitoring or alerting mechanisms, it provides a useful audit trail for user interaction with the model.