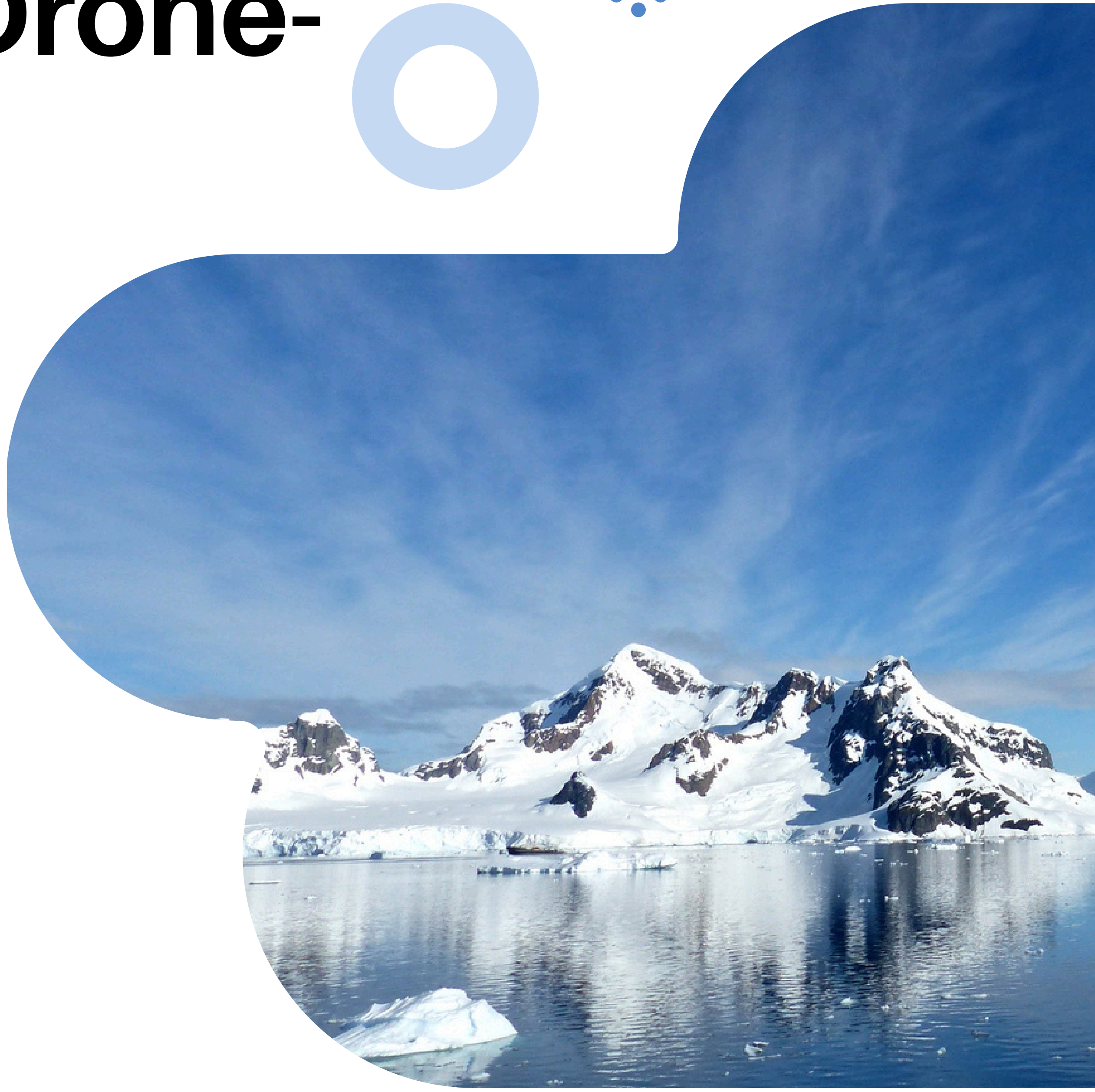
Intelligent IIoT Service Deployment using MEC and Federated Learning in Drone-Assisted Smart Agriculture

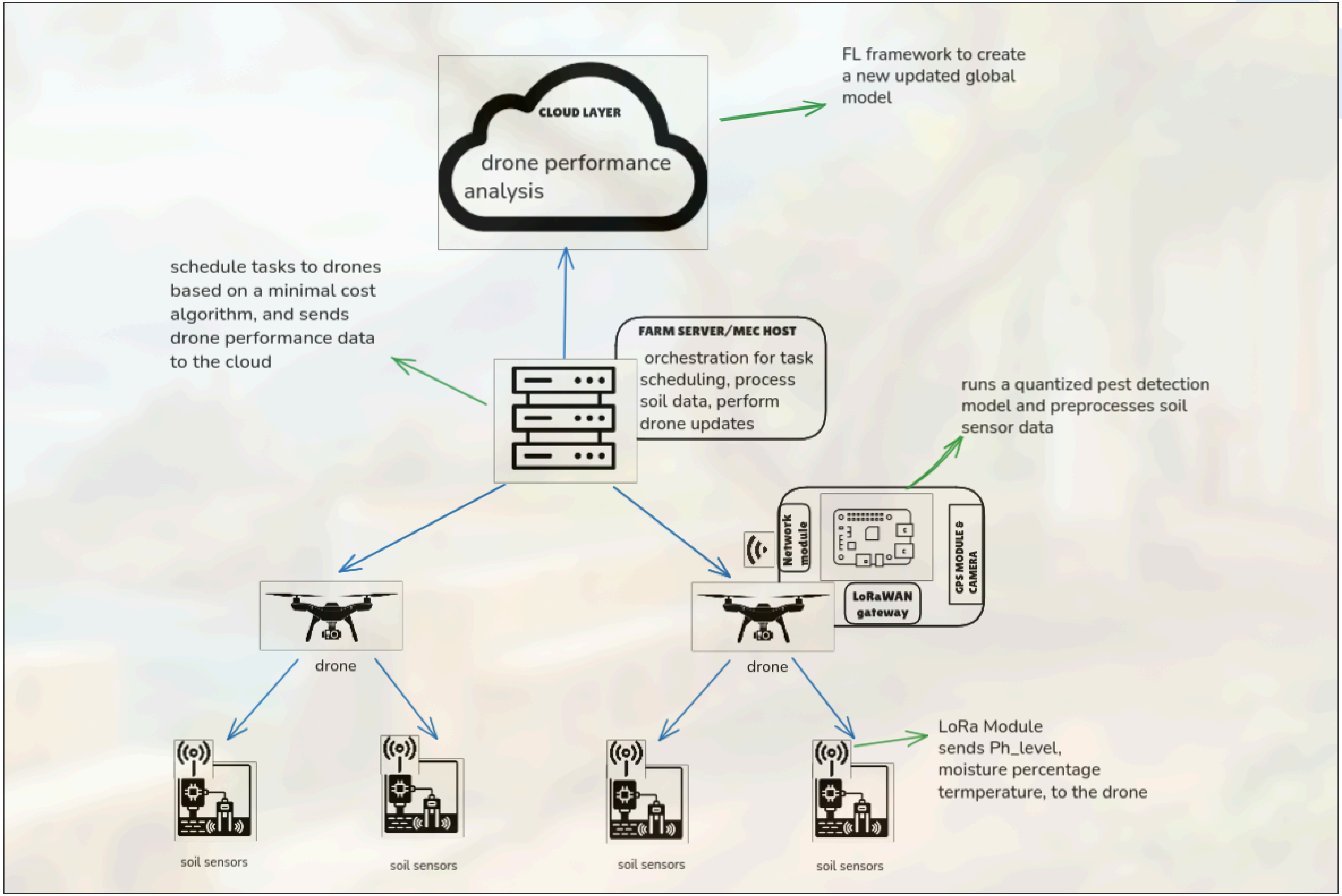


Team No. 16
EDGLERS

PROBLEM STATEMENT

- Modern farming faces challenges in rapidly deploying and managing digital services like crop monitoring, pest detection, and irrigation control.
- Traditional cloud-based or manual IT setups cause high delay (latency), network congestion, and integration obstacles—especially over vast or remote fields where reliable connectivity is critical.
- Manual or slow deployment makes it impossible to react to sudden changes (like weather or pests) and to scale up for large harvest operations.

Edge Architecture



Topology & Modules

Topology Design

- Cloud (Node 0): powerful central server (analytics).
- MEC/Fog (Node 1): local farm server, mid-compute, low latency.
- Drones (Nodes 50+): mobile collectors, Wi-Fi & mesh links.
- Soil Sensors (Nodes 100+): low-power IoT devices, LoRaWAN links + backup to MEC.

Applications/Modules

- SensorDataCollection: sensors create soil readings.
- Imagecaputure: takes photos from csv file and feeds it to the model
- DroneProcessing: drones process & forward data to MEC.
- CloudAnalytics: MEC forwards aggregated data to cloud.
- Placement:
  - Sensors → sensor nodes
  - Drones → drone nodes
  - MEC → node 1
  - Cloud → node 0

```python
def scheduleTask(self, sensorId, currTime):
    sensorPos = self.sensorPositions[sensorId]
    bestDrone, minCost = None, float('inf')

    for droneId, dronePos in self.dronePositions.items():
        dist = self.calculateDistance(dronePos, sensorPos)
        batPenalty = (100 - self.droneBattery[droneId]) * 0.5
        cost = dist + batPenalty

        if self.droneBattery[droneId] > 20 and cost < minCost:
            minCost, bestDrone = cost, droneId

    if bestDrone is not None:
        self.assignedTasks[bestDrone].append({...})
        self.dronePositions[bestDrone] = sensorPos
        # Decrease battery (min 5 units, or remaining battery if <5)
        self.droneBattery[bestDrone] -= min(5, self.droneBattery[bestDrone])
        return bestDrone

    return None
```

Task Scheduling & Simulation Logic

Task Scheduling (Greedy Algorithm)

- Cost = distance(drone, sensor) + 0.5 × (100 − battery%)
- Only drones with >20% battery considered.
- Best drone chosen → moves to sensor, collects data, battery −5%.
- If battery <30% → upload data to MEC & recharge (+20%).

Simulation Logic

- Event generation:
  - Sensors → every 300 units,Drones → every 250 units
  - MEC → sends to cloud every 800 units
- Custom Strategy: runs periodically to assign drones dynamically.
- Outputs (CSV):
  - Soil sensor data,Drone collection logs,MEC processing results,YAFS trace for latency & paths

Questions from Review one

1.How does the proposed solution enable fast reaction to sudden environmental changes?

Drones perform local training on new data, enabling rapid self-evolution and immediate adaptation to localized environmental changes without waiting for central commands.

2.How do you scale when multiple drones/sensors are working together?

The system scales efficiently as each Drone Client works independently, with the server only collecting model weights. Asynchronous Federated Averaging avoids bottlenecks, maintaining performance as the fleet grows.

3.What happens if one drone or MEC node fails—does the whole system stop?

The system is resilient because its decentralized design lets other drones keep working even if one fails. The aggregation script uses available updates, so the global model keeps improving.

Changes from review 1

1.Changes the LoRaWAN receiver from independent drones to the MEC server

2.Added an FL framework that updates the drone models every month

Tasks left for review 3

1.Performance analysis yet to be implemented on the cloud server

2.Complete model and architecutre integration

3.Yeild prediction algorithms and models should be implemented

YAFS Simulation Framework Implementation

01 Simulation Components

- **Topology Builder:**Creates hierarchical network with nodes (sensors, drones, MEC, cloud) and edges (LoRaWAN, WiFi, fiber connections)
- **Application Modeling**: Defines data flow from sensors → drones → MEC → cloud with realistic message sizes and processing requirements
- **Resource Management**: Simulates CPU, RAM, storage, and network bandwidth usage across all nodes

02 YAFS Features Utilized

- **Dynamic Message Routing**: Realistic network latency and bandwidth simulation\
- **Resource Monitoring**: Real-time tracking of node utilization and performance metrics
- **Event-Driven Simulation**: Time-stepped execution modeling actual network behavior

CNN Model For Pest Detection

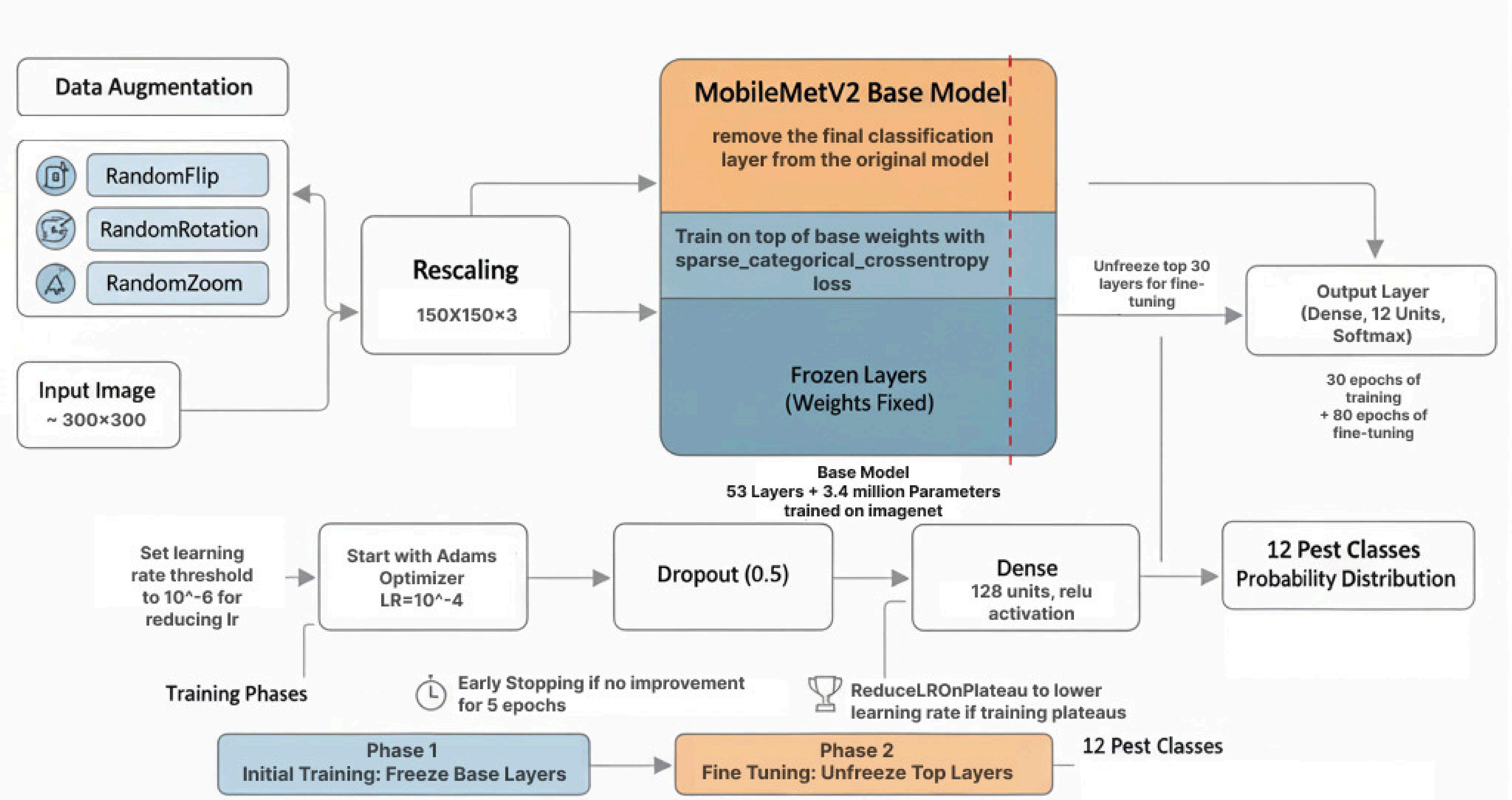
Initial Attempt: Built a CNN from scratch:

- Resized images (150×150×3), 80-20 split, data augmentation
- Architecture: Conv2D + MaxPooling → Dense layers → Softmax (12 pest classes)
- Trained with Adam optimizer, sparse categorical cross-entropy
- Achieved ~50% accuracy with ~60MB model (20–30 epochs, early stopping)

Second Iteration: Applied Transfer Learning with MobileNetV2:

- Used pretrained ImageNet weights, removed top layers
- Added Global Average Pooling, Dropout(0.3), Dense(12, Softmax)
- Used callbacks: Early Stopping + Reduce LR on Plateau
- Fine-tuned by unfreezing last 30 layers, retrained with lower LR
- Achieved ~85% accuracy with ~110MB model (110 epochs, early stopping)

System Architecture CNN Model



Federated Learning Model

Built a Federated Learning system that trains a new neural network taking weights from CNN models from drone fleet, improving the model's pest detection accuracy.

How It Works

- Initially a central server distributes a global Neural Network to drone clients.
- Each drone improves the model locally on its private data.
- Drones upload only their small model weight updates, which the server combines using Federated Averaging (FedAvg) to create an improved global model.

Designed a 3-part FL simulation that trains a model across distributed devices, solving the key challenge of network efficiency.

Result & Verification:
The final model achieved ~98% accuracy on a test dataset, proving the FL system is a high-performance, privacy-preserving solution.

Light-weight Fuzzy Logic for Soil pH Classification

Objective

- To build a lightweight, interpretable CI model that can run on a low-power microcontroller for real-time, on-site soil analysis.

How It Works

- The model uses a Fuzzy Inference System with a simple "if-then" rule base.
- It takes a precise pH value from a sensor and classifies it into one of three linguistic categories: Acidic, Neutral, or Alkaline.
- This approach demonstrates a rule-based CI technique, which is highly efficient and easy to understand.

The final working script achieved ~97% accuracy in classifying soil pH.

Future Work: Build a Adaptive Neuro-Fuzzy Inference System (ANFIS)

To build a hybrid CI model that combines the learning capabilities of a Neural Network with the transparent, rule-based structure of a Fuzzy Logic system

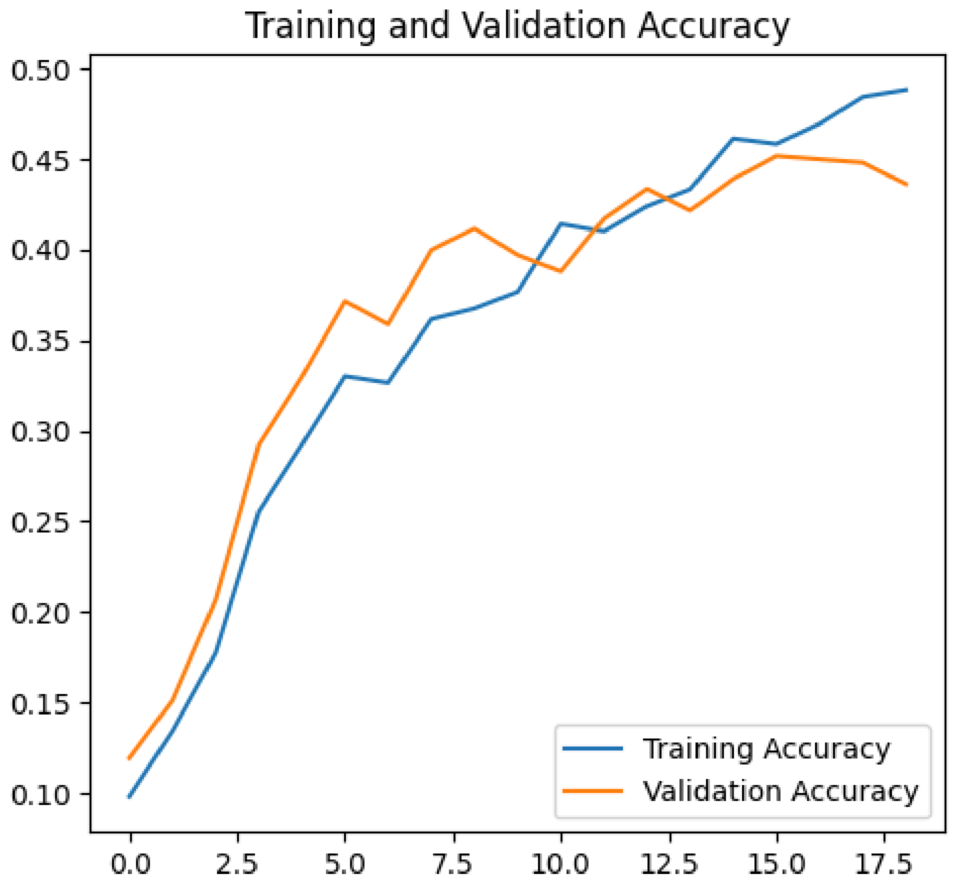
```
[starkiller@starkiller-linux fl_model]$ python drone_client.py
2025-09-18 11:02:41.603617: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn t
, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-09-18 11:02:41.669858: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'

--- Starting Local Training on drone_2744 ---
/usr/lib/python3.13/site-packages/keras/src/saving/saving_lib.py:797: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 variables whereas the saved optimizer has 74 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
Successfully loaded global weights from global_model_initial.weights.h5
Simulating collection of new local data...
Found 30 files belonging to 3 classes.
Starting local fine-tuning...
Epoch 1/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 13s 129ms/step - accuracy: 0.1667 - loss: 3.5472
Epoch 2/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step - accuracy: 0.1333 - loss: 3.3055
Epoch 3/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step - accuracy: 0.1667 - loss: 2.8302
Epoch 4/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 0.2000 - loss: 3.0405
Epoch 5/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step - accuracy: 0.0667 - loss: 3.0590
Local fine-tuning complete.
Saved updated weights to drone_2744_update_1758173583.weights.h5
```

```
[starkiller@starkiller-linux fl_model]$ zsh
~/Documents/fl_model ❯ python monthly_aggregation_script.py
2025-09-18 11:01:55.623351: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn t
, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-09-18 11:01:55.689725: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
--- Starting Monthly Federated Averaging ---
Using latest global model as base: global_models/global_model_initial.weights.h5
Found 3 updates to process.
/usr/lib/python3.13/site-packages/keras/src/saving/saving_lib.py:797: UserWarning: Skipping variable loading for optimizer 'rmsprop', because it has 2 variables whereas the saved optimizer has 74 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
Aggregating weights from 3 clients, with a total of 90 new samples.
--- Aggregation Complete ---
New global model saved as: global_models/global_model_2025-09-18.weights.h5
Moved processed updates to drone_updates/processed_2025-09-18
```

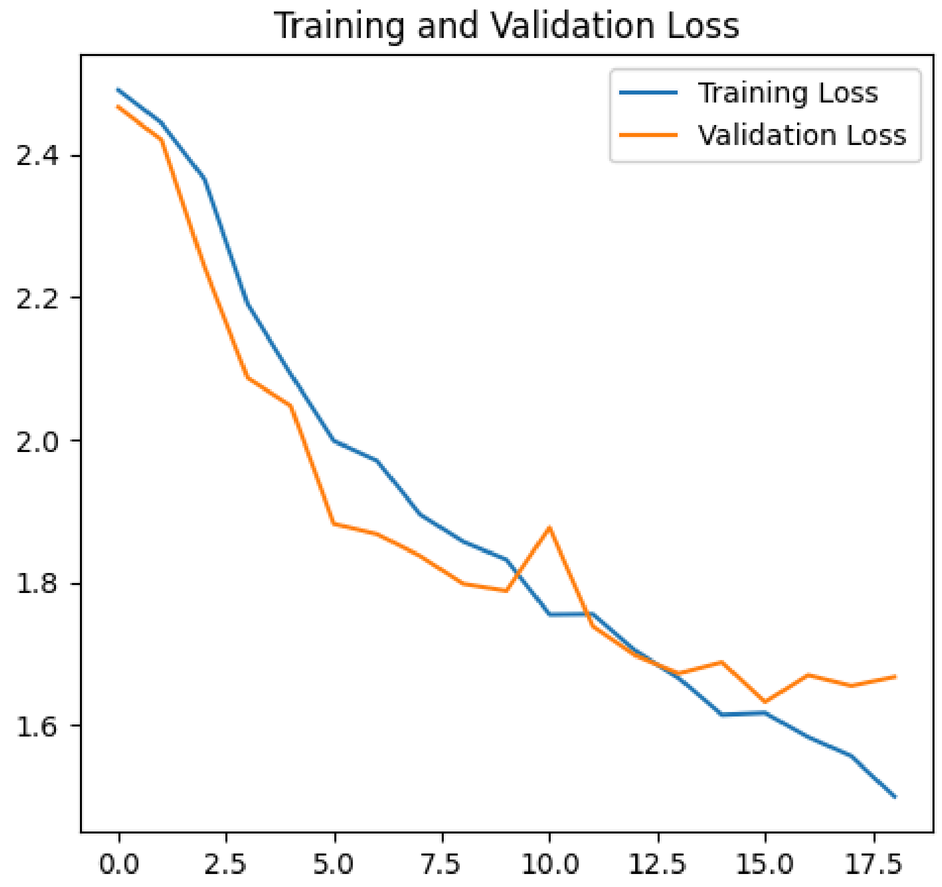
THANK YOU!

www.reallygreatsite.com

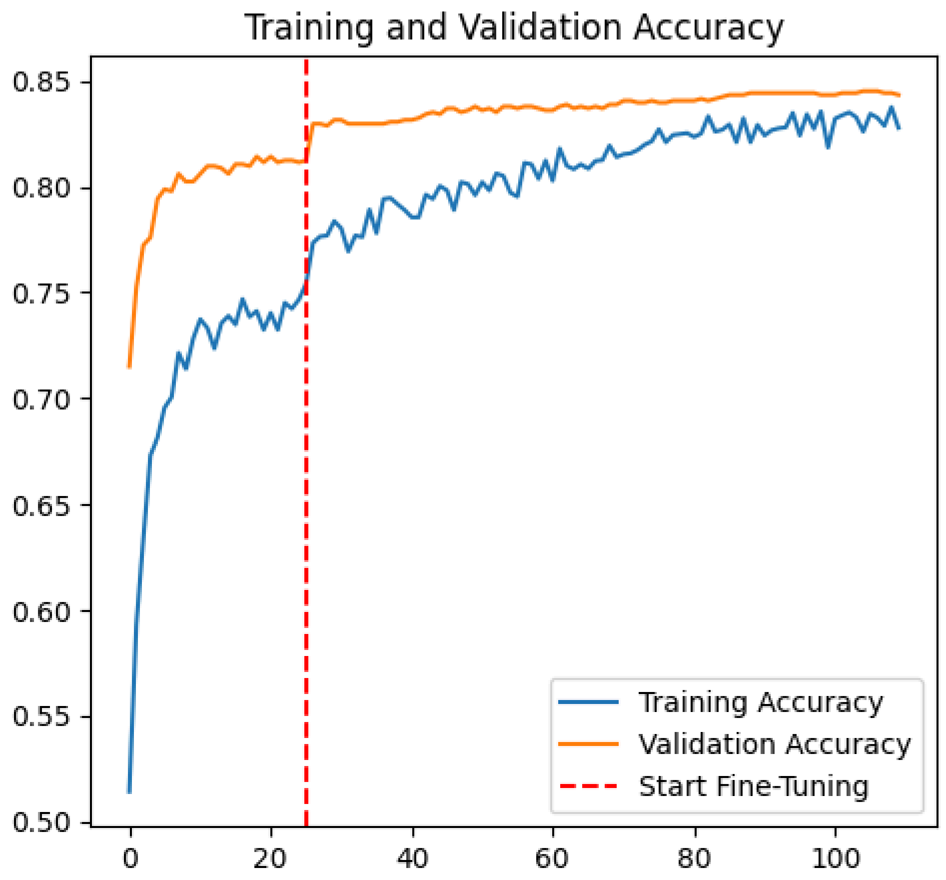
Results



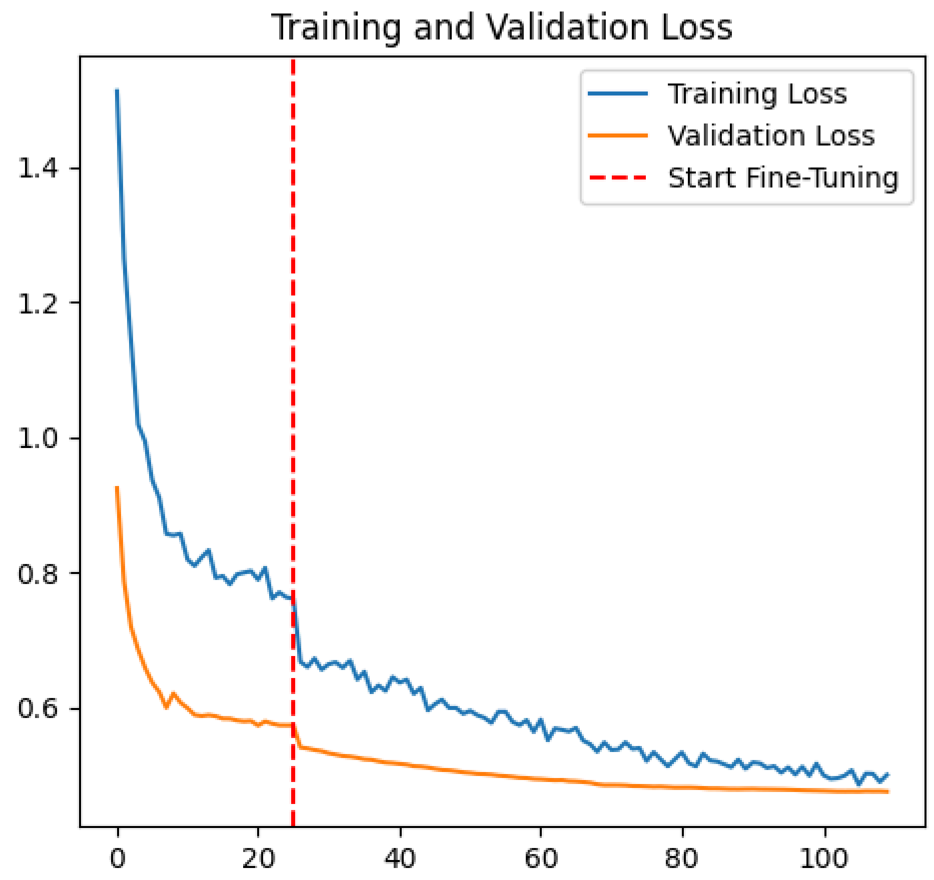
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.50 0.45 0.40 0.35 0.30 0.25 0.20 0.15 0.10
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5



Training and Validation Loss
Training Loss
Validation Loss
2.4 2.2 2.0 1.8 1.6
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5



Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Start Fine-Tuning
0.85 0.80 0.75 0.70 0.65 0.60 0.55 0.50
0 20 40 60 80 100



Training and Validation Loss
Training Loss
Validation Loss
Start Fine-Tuning
1.4 1.2 1.0 0.8 0.6
0 20 40 60 80 100