Advanced Methodology for Optimizing Lightweight Deep Neural Networks in Citrus Disease Detection

Anonymous

March 8, 2025

Contents

1 Introduction

This document presents an in-depth, state-of-the-art methodology for adapting and training four lightweight deep neural network architectures—Tiny YOLO, EfficientNet, MobileNetV2, and ShuffleNet—for citrus disease detection in uncontrolled agricultural environments. The methodology covers the complete pipeline from data acquisition and preprocessing through model-specific architectural adaptations, loss formulation, optimization strategies, and mobile deployment techniques. Emphasis is placed on leveraging advanced techniques for robust generalization, rapid convergence, and resource-efficient inference on mobile devices.

2 Data Acquisition, Preprocessing, and Augmentation

2.1 Data Collection and Standardization

Our dataset comprises high-resolution images capturing both healthy and diseased citrus specimens under varied field conditions (e.g., variable lighting, occlusions, and complex backgrounds). To ensure consistent input across the different models, each image is resized according to the target architecture requirements (e.g., 416×416 for Tiny YOLO and 224×224 for CNN-based models). A critical preprocessing step is the standardization of raw pixel intensities. Given an input image I with pixel values x, we perform standard score normalization as follows:

$$\hat{x} = \frac{x - \mu}{\sigma}, \tag{1}$$

where μ and σ denote the mean and standard deviation computed over the entire training dataset. This normalization stabilizes the input distribution, accelerates convergence, and prevents numerical instabilities during training.

2.2 Augmentation: Geometric and Photometric Enhancements

Robustness to real-world variations is achieved through an extensive data augmentation pipeline that includes both geometric and photometric transformations. The key augmentation techniques are:

- **Rotation:** Each image is randomly rotated by an angle θ (in radians). The new coordinates (x', y') are computed via:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2}$$

- **Scaling:** Uniform scaling is applied to adjust the image size by a factor s:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{3}$$

  where s is selected from a predefined range to simulate varying camera distances.

- **Flipping and Cropping:** Random horizontal and vertical flips are applied with fixed probabilities to capture symmetric features. Additionally, random cropping and padding simulate off-center captures.

- **Photometric Adjustments:** The pipeline introduces random adjustments to brightness, contrast, and saturation. Histogram equalization and adaptive gamma corrections are used to mitigate variable illumination, ensuring that the model learns illumination-invariant features.

- **Noise Injection:** Gaussian noise and salt-and-pepper noise are optionally introduced to mimic sensor imperfections and environmental artifacts.

3 Model-Specific Methodologies and Architectural Adaptations

Each model is carefully adapted for the citrus disease detection task. Although all models share a common preprocessing pipeline, each undergoes specialized modifications in its top layers, activation functions, and training strategies to address computational constraints and the unique challenges presented by field data.

3.1 Tiny YOLO Adaptation for Citrus Classification

Tiny YOLO, originally designed for real-time object detection, is repurposed for fine-grained disease classification via significant modifications to its detection head.

3.1.1 Preprocessing and Input Adaptation

Tiny YOLO requires input images of size 416×416 pixels. In addition to applying the normalization (Equation 1) and geometric augmentations (Equations 2 and 3), an optional color-space transformation (e.g., RGB-to-HSV) is applied to better capture hue variations associated with specific disease symptoms.

3.1.2 Architectural Modifications

Key modifications to the original Tiny YOLO architecture include:

- **Activation Function:** A leaky ReLU is used to avoid the dead neuron problem:
$$f(x) = \max(0.1x, x), \tag{T4}$$
ensuring a small gradient is preserved for negative inputs.

- **Revised Spatial Encoding:** Traditional YOLO predicts bounding box centers and dimensions using:
$$b_x = \sigma(t_x) + c_x, \tag{T5}$$
$$b_w = p_w \cdot e^{t_w}, \quad b_h = p_h \cdot e^{t_h}, \tag{T6}$$
which, in our classification task, inspire enhanced spatial feature mapping prior to global pooling.

- **Custom Classification Head:** The final convolutional layers are replaced by a fully connected network that directly outputs class probabilities. The transformation is defined as:
$$z = W_f \cdot F + b_f, \tag{T7}$$
where F is the vectorized convolutional feature map, and W_f, b_f are learnable parameters for C classes.

- **Softmax Normalization:** The logits are converted to probabilities via the softmax function:
$$p(c|x) = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}}, \tag{T8}$$
ensuring a valid probability distribution.

- **Dropout Regularization:** Dropout is applied to mitigate overfitting:
$$\tilde{F} = F \odot M, \tag{T9}$$
where M is a binary mask sampled from a Bernoulli distribution with a tuned keep probability.

3.1.3 Loss Formulation and Optimization Strategy

The training objective for Tiny YOLO is defined by a composite loss function:

- **Categorical Cross-Entropy Loss:**
$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(p_i), \tag{T10}$$
where y_i is the one-hot encoded ground truth.

- **L2 Regularization:**
$$\mathcal{L}_{reg} = \lambda \sum_{j} w_j^2, \tag{T11}$$
with λ being a hyperparameter controlling the regularization strength.

- **Total Loss:**
$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg}, \tag{T12}$$
which is minimized during training.

- **Cosine Annealing Learning Rate:**
$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{\pi t}{T}\right)\right), \tag{T13}$$
where T is the total number of training epochs.

- **Adaptive Optimization with Adam:**
$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{T14}$$
where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates.

- **Model Compression via Pruning:**
$$w' = \begin{cases} w, & \text{if } |w| \geq \tau, \\ 0, & \text{if } |w| < \tau, \end{cases} \tag{T15}$$
where τ is the pruning threshold.

3.2 EfficientNet Adaptation with Compound Scaling and SE Blocks

EfficientNet leverages a compound scaling method to uniformly scale the network's depth, width, and resolution. We adapt EfficientNet-B0 as a baseline and incorporate additional modifications for citrus disease classification.

3.2.1 Compound Scaling and Preprocessing

EfficientNet uses a compound coefficient ϕ to adjust network dimensions:

$$d = \alpha^{\phi}, \quad w = \beta^{\phi}, \quad r = \gamma^{\phi}, \tag{E4}$$

subject to the constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \tag{E5}$$

which guarantees balanced expansion across all dimensions. Standard normalization (Equation 1) and the augmentation pipeline (Equations 2 and 3) are applied to all inputs.

3.2.2 Architectural Enhancements

EfficientNet integrates squeeze-and-excitation (SE) blocks to recalibrate channel-wise feature responses:

$$s = \sigma(W_2 \cdot \delta(W_1 \cdot z)), \tag{E6}$$

where z is obtained via global pooling of feature maps, W_1 and W_2 are trainable parameters, δ represents the ReLU activation, and σ is the sigmoid function.

Additional modifications include:

- **Global Average Pooling (GAP):**

$$F = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} f(i,j), \tag{E7}$$

  reducing the spatial dimensions.

- **Custom Fully Connected Layer:**

$$z = W_f \cdot F + b_f, \tag{E8}$$

  mapping the pooled features to the classification logits.

- **Softmax Transformation:**

$$p(c|x) = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}}, \tag{E9}$$

  which converts logits to class probabilities.

- **Dropout Regularization:**

$$\tilde{F} = F \odot M, \tag{E10}$$

  where M is a dropout mask.

3.2.3 Loss and Optimization

The training objective is defined through:

- **Cross-Entropy Loss:**

$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(p_i), \tag{E11}$$

- **L2 Regularization Loss:**

$$\mathcal{L}_{reg} = \lambda \sum_{j} w_j^2, \tag{E12}$$

- **Total Loss:**

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg}, \tag{E13}$$

- **Adam Optimizer Update:**

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{E14}$$

- **Cosine Annealing Learning Rate:**

$$\eta_t = \eta_{\min} + \frac{1}{2}\left(\eta_{\max} - \eta_{\min}\right)\left(1 + \cos\left(\frac{\pi t}{T}\right)\right), \tag{E15}$$

which together ensure balanced convergence and generalization.

3.3 MobileNetV2: Depthwise Separable Convolutions and Inverted Residuals

MobileNetV2 is designed for efficiency using depthwise separable convolutions and an inverted residual structure.

3.3.1 Architectural Overview and Preprocessing

MobileNetV2 accepts 224×224 pixel inputs. The preprocessing pipeline (Equation 1 for normalization and Equations 2 and 3 for augmentation) is applied uniformly.

3.3.2 Core Convolutional Modules

- **Depthwise Convolution:** Each filter is applied to a single input channel:

$$y[i,j,c] = \sum_{m,n} x[i+m, j+n, c] \cdot k[m,n,c], \tag{M4}$$

  where x is the input and k is the convolutional kernel.

- **Pointwise Convolution:** A subsequent 1×1 convolution aggregates the outputs:

$$z[i,j,d] = \sum_{c} y[i,j,c] \cdot w[c,d], \tag{M5}$$

  where w is the weight matrix mapping to the new feature space.

- **Inverted Residual Block:** The input is first expanded by a factor t:

$$F_{exp} = \phi(F_{in}, t), \tag{M6}$$

  followed by depthwise convolution and a linear projection.

3.3.3 Tailored Classification Head

The classification head is re-engineered as follows:

- **Global Average Pooling (GAP):**

$$F = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} f(i,j), \tag{M7}$$

- **Fully Connected Transformation:**

$$z = W_f \cdot F + b_f, \tag{M8}$$

- **Softmax Activation:**

$$p(c|x) = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}}, \tag{M9}$$

- **Dropout Regularization:**

$$\tilde{F} = F \odot M, \tag{M10}$$

  where M is the dropout mask.

3.3.4 Loss Formulation and Learning Rate Adaptation

The training objectives are:

- **Categorical Cross-Entropy Loss:**

$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(p_i), \tag{M11}$$

- **L2 Regularization:**

$$\mathcal{L}_{reg} = \lambda \sum_{j} w_j^2, \tag{M12}$$

- **Total Loss:**

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg}, \tag{M13}$$

- **Cosine Annealing Learning Rate:**

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right), \tag{M14}$$

- **Adam Optimizer Update Rule:**

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{M15}$$

ensuring competitive performance with minimal computational overhead.

3.4 ShuffleNet: Group Convolutions and Channel Shuffle Mechanism

ShuffleNet is optimized for low-resource devices by exploiting group convolutions and a unique channel shuffle operation.

3.4.1 Preprocessing and Input Preparation

ShuffleNet uses input images resized to 224×224 pixels. Standard normalization (Equation 1) and augmentation (Equations 2 and 3) are applied uniformly.

3.4.2 Core Architectural Innovations

Key innovations include:

- **Group Convolution:** Input channels are partitioned into groups, with each group convolved independently:

$$y_g = x_g * k_g, \tag{S4}$$

where x_g and k_g represent the inputs and kernels for group g.

- **Channel Shuffle:** To promote inter-group information flow, a permutation matrix P shuffles the channels:

$$X_{\text{shuffled}} = PX, \tag{S5}$$

- **Bottleneck Structure:** A bottleneck layer reduces channel dimensionality while preserving critical features:

$$F_{bottleneck} = \psi(F_{in}, r), \tag{S6}$$

where r is the reduction ratio.

3.4.3 Modified Classification Head

The final layers are modified as follows:

- **Global Average Pooling (GAP):**

$$F = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} f(i,j), \tag{S7}$$

- **Fully Connected Layer:**

$$z = W_f \cdot F + b_f, \tag{S8}$$

- **Softmax Activation:**

$$p(c|x) = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}}, \tag{S9}$$

- **Dropout Regularization:**

$$\tilde{F} = F \odot M, \tag{S10}$$

ensuring that the network does not overfit.

3.4.4 Loss Function and Training Dynamics

The training regimen for ShuffleNet includes:

- **Categorical Cross-Entropy Loss:**

$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(p_i), \tag{S11}$$

- **L2 Regularization Loss:**

$$\mathcal{L}_{reg} = \lambda \sum_{j} w_j^2, \tag{S12}$$

- **Total Loss:**

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg}, \tag{S13}$$

- **Cosine Annealing Learning Rate:**

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{\pi t}{T}\right)\right), \tag{S14}$$

- **Adam Optimizer Update Rule:**

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{S15}$$

which together guarantee effective convergence and deployment readiness.

4 Unified Optimization and Training Framework

Although each model features unique architectural characteristics, our training framework employs several shared principles:

4.1 Standardized Preprocessing and Augmentation

All models begin with:

- **Normalization:** As per Equation (1).
- **Data Augmentation:** Using the geometric (Equations 2 and 3) and photometric transformations.

This standardization ensures uniformity of input data and allows for fair model comparisons.

4.2 Composite Loss Function and Regularization

Each model minimizes a loss combining:

- **Categorical Cross-Entropy Loss:** (Equations T10, E11, M11, S11)
- **L2 Regularization:** (Equations T11, E12, M12, S12)

Thus, the total loss is defined as:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg}. \tag{1}$$

4.3 Adaptive Learning Rate Scheduling

A cosine annealing schedule (Equations T13, E15, M14, S14) gradually decreases the learning rate, promoting smooth convergence and fine-grained updates in later epochs.

4.4 Optimization with Adam

All models are optimized using the Adam optimizer (Equations T14, E14, M15, S15) which adjusts the learning rate per parameter and effectively handles sparse gradients.

4.5 Model Compression Techniques

To meet the constraints of mobile devices, we employ:

- **Magnitude-Based Pruning:** (Equation T15)
- **Quantization:** Post-training quantization (e.g., 32-bit to 8-bit conversion)
- **Future Direction:** Knowledge distillation to further compress models.

5 Model Comparison and Deployment Considerations

5.1 Evaluation Metrics

Performance is rigorously evaluated on a held-out test set using standard metrics:

- **Accuracy:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

- **Precision:**

$$\text{Precision} = \frac{TP}{TP + FP},$$

- **Recall:**

$$\text{Recall} = \frac{TP}{TP + FN},$$

- **F1-Score:**

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

5.2 Real-Time Inference on Mobile Devices

To ensure deployability on smartphones:

- **Conversion:** Models are converted to mobile-friendly formats such as TensorFlow Lite or ONNX.
- **Pipeline Optimization:** Quantization and pruning are applied to reduce latency and memory usage.
- **Inference Time:** Effective per-image inference time is computed as:

$$t_{\text{eff}} = \frac{T_{\text{batch}}}{N}, \tag{2}$$

where T_{batch} is the total time for a batch of N images.

5.3 Comparative Analysis

The unified framework enables direct comparisons across models in terms of:

- **Accuracy and Generalization**
- **Computational Efficiency:** Model size, latency, and energy consumption.
- **Scalability and Robustness:** Performance under varying field conditions.

6 Advanced Considerations and Future Directions

While the methodology described herein provides a robust foundation, several avenues for further enhancement remain:

6.1 Knowledge Distillation and Ensemble Methods

Future work may incorporate:

- **Teacher-Student Frameworks:** A larger teacher network can guide the training of a compact student network.
- **Ensemble Techniques:** Combining predictions from multiple architectures to improve robustness.

6.2 Adaptive Data Augmentation Strategies

Further improvements in data augmentation might include:

- **Curriculum Learning:** Gradually increasing augmentation complexity based on model confidence.
- **Online Augmentation Policies:** Employing reinforcement learning to dynamically adjust augmentation parameters.

6.3 Neural Architecture Search (NAS)

Incorporating hardware-aware NAS can:

- Optimize architectural parameters (depth, width, and resolution) for specific mobile constraints.
- Enhance the trade-off between accuracy and inference latency.

6.4 Real-Time Monitoring and Continual Learning

Given the dynamic nature of field conditions, future work may include:

- **On-Device Adaptation:** Continual learning techniques to update model parameters in response to new data.
- **Edge-Cloud Collaboration:** Periodic synchronization between devices and central servers for model refinement.

6.5 Hardware-Specific Optimizations

Additional efficiency gains can be achieved by:

- Exploiting mobile-specific hardware accelerators (e.g., DSPs, NPUs).
- Employing software-hardware co-design to fully leverage device capabilities.

7 Conclusion

In this document, we have detailed a comprehensive, technically rigorous methodology for adapting and training four lightweight deep learning models—Tiny YOLO, EfficientNet, MobileNetV2, and ShuffleNet—for real-time citrus disease detection. By integrating advanced data preprocessing, customized architectural modifications, robust loss formulations, adaptive learning rate schedules, and state-of-the-art optimization strategies, we provide a reproducible and scalable framework suitable for deployment on resource-constrained mobile devices.

Key highlights include:

- **Unified Preprocessing and Augmentation:** Consistent data input across all architectures.
- **Model-Specific Adaptations:** Detailed modifications tailored to each architecture.
- **Rigorous Optimization Framework:** Use of composite loss functions, cosine annealing schedules, and Adam optimizer.
- **Deployment Readiness:** Incorporation of model compression and quantization techniques.

This exhaustive framework not only addresses the challenges of deploying deep learning in agricultural settings but also provides a solid foundation for future enhancements. By leveraging strategies such as knowledge distillation, adaptive augmentation, NAS, and hardware-specific optimizations, researchers can further refine this methodology to tackle emerging diseases and evolving field conditions. Ultimately, the techniques presented here push the boundaries of practical deep learning applications in precision agriculture, contributing to improved crop health monitoring and sustainable farming practices.

End of Document