MDPI

Article

A Memory Efficient Two-Stage Pipeline for On-Device Plant Disease Segmentation and Classification at the Edge

Kiran K. Kethineni[1,*], Azalea Tang[2], Saraju P. Mohanty[3] and Elias Kougianos[4]

[1] *Dept. of Computer Science and Engineering, University of North Texas; kirankumarkethineni@my.unt.edu*
[2] *Dept. of Computer Science and Engineering, University of North Texas; azaleatang@my.unt.edu*
[3] *Dept. of Computer Science and Engineering, University of North Texas; smohanty@ieee.org*
[4] *Dept. of Electrical Engineering, University of North Texas; elias.kougianos@unt.edu*
* Correspondence: kirankumarkethineni@my.unt.edu

Abstract

Plant disease identification on resource constrained edge devices must satisfy two conflicting goals: a detailed enough perception of the leaf to localize affected tissue for variable rate spraying and a strict memory / compute budget. Single network solutions that jointly learn pixel wise masks and fine grained disease labels tend to oversubscribe RAM and Flash on resource constrained edge hardware. This paper proposes a two stage edge pipeline that decouples the two subproblems. Stage 1 is a compact fully convolutional segmentation network that ingests the R, G, and B channels jointly and produces a binary "diseased vs. background" mask at full input resolution. It uses Separable-Conv2D blocks with an *addition based* Feature Pyramid Network (FPN) [1] decoder and a single 8× nearest neighbor upsampling at the head, eliminating the heavy transpose convolution decoder that normally dominates edge memory. Stage 2 is the previously published Chroma-Sense classifier [2], which processes the same RGB frame independently to produce a disease label. Critically, the pipeline's target metric is not pixel level segmentation accuracy but *spray decision accuracy*: the segmentation mask is converted to a per plant severity ratio, which is thresholded to produce a binary spray or skip command. We show theoretically that this decision is robust to segmentation errors up to ∼10% for plants not near the threshold boundary, a property we call *task sufficient accuracy*. Because the sprayer itself actuates at cell level rather than pixel level granularity, the "blobbing" introduced by the lightweight decoder design is tolerated as controlled foreground overcoverage. We evaluate both pipeline stages on the PlantSeg dataset [3], using a curated subset of 10 plant species covering 34 disease classes. Stage 1 achieves a mean foreground recall of 0.889 across all species (per species range 0.86 to 0.92), with foreground IoU of 0.62 to 0.73, a gap expected from the coarse upsampling design. Stage 2 achieves a mean classification accuracy of 0.871 (macro F1: 0.859), exceeding 0.90 on data rich crops (Apple, Cherry, Corn) and reflecting data scarcity on Banana and Soybean.

Keywords: edge computing; TinyML; semantic segmentation; FPN; separable convolution; precision agriculture; variable rate spraying; Chroma-Sense; plant disease detection

Received:
Accepted:
Published:

1. Introduction

Smart agriculture systems increasingly aim to close the loop between perception ("where is the disease?") and actuation ("spray only there") on the same embedded device. Running a convolutional neural network (CNN) on a microcontroller or a single

board computer, rather than on a remote cloud, removes the network roundtrip and the dependency on rural connectivity [4–6]. The penalty is that edge hardware has heap memory in the hundreds of KBs and Flash memory in the low MBs, which is an order of magnitude below what common encoder decoder segmentation models consume.

Previous work has approached the problem along two axes. One line of research reduces the per parameter cost of inference using depthwise separable convolutions, inverted residuals, and squeeze and excitation modules [7–10]. A second line of work targets plant disease specific structure, arguing that the visual semantics of lesions (spots, rings, mosaics, powdery patches) are simpler than those of general object recognition and can be encoded with shallow networks [11,12]. Our earlier Chroma-Sense classifier [2] fits into the second camp: it processes the R, G, and B channels of a cropped leaf independently using a weight shared feature extractor, reducing Flash footprint to 54 KB while achieving competitive accuracy on the PlantVillage benchmark [13].

Chroma-Sense [2], however, is a pure classifier. It produces a disease label from an input image but generates no spatial mask. In a real spraying context, the upstream image typically contains several leaves, stems, and soil; the actuator needs a pixel wise or cell wise indication of which parts of the frame to treat. Adding a full segmentation decoder on top of Chroma-Sense pushes the combined model past the memory envelope of the very devices it targets.

This paper proposes a two stage pipeline that splits the problem along its natural seam. Stage 1 is a lightweight fully convolutional network that learns a two class, diseased versus background, mask from the joint RGB input. The task is fundamentally easier than multi class classification: the model only needs to separate lesion texture from healthy leaf and canopy, regardless of which disease is present. Because of that, Stage 1 can afford an aggressively pruned decoder that relies on element wise addition rather than concatenation in its skip connections, fuses features at two scales only, and finishes with a single $8\times$ nearest neighbor upsampling. Stage 2 is the published Chroma-Sense classifier [2], which processes the same RGB input independently to identify the specific disease when that information is required downstream (to select a herbicide or to log the infection for disease spread analytics).

A key design choice is to accept the coarse, "blobby" output of the lightweight decoder model rather than fight it. The downstream actuator in our target application is a variable rate agricultural sprayer, whose nozzle footprint is already several centimeters wide and whose actuation is commanded at a coarse spatial grid. A mask that slightly overcovers the true lesion does not translate into visible overspray in the field, and it removes the need for the boundary refinement layers that dominate the memory budget in models such as U-Net [14] and DeepLab [15]. Once the per cell disease decision produced by this pipeline is available, a separate route planning layer, such as SprayCraft [16], can optimize the spray path across the field; that downstream problem is outside the scope of the present work.

Why frugality is the binding constraint.

Agricultural edge hardware operates under tight memory and compute constraints: available Flash is in the low hundreds of kilobytes after model storage, and peak activation SRAM is comparably limited. A segmentation decoder that oversubscribes these budgets cannot be deployed, making frugal architecture a deployment prerequisite rather than a performance optimization. The design space therefore narrows to techniques that can trade accuracy *informationally* for memory: reducing the amount of information the model must produce (binary instead of multi class), reducing the precision of its spatial output (coarse instead of fine), or staggering the model's workload over time (gated classifier rather than always on). The pipeline proposed here uses all three levers at once.

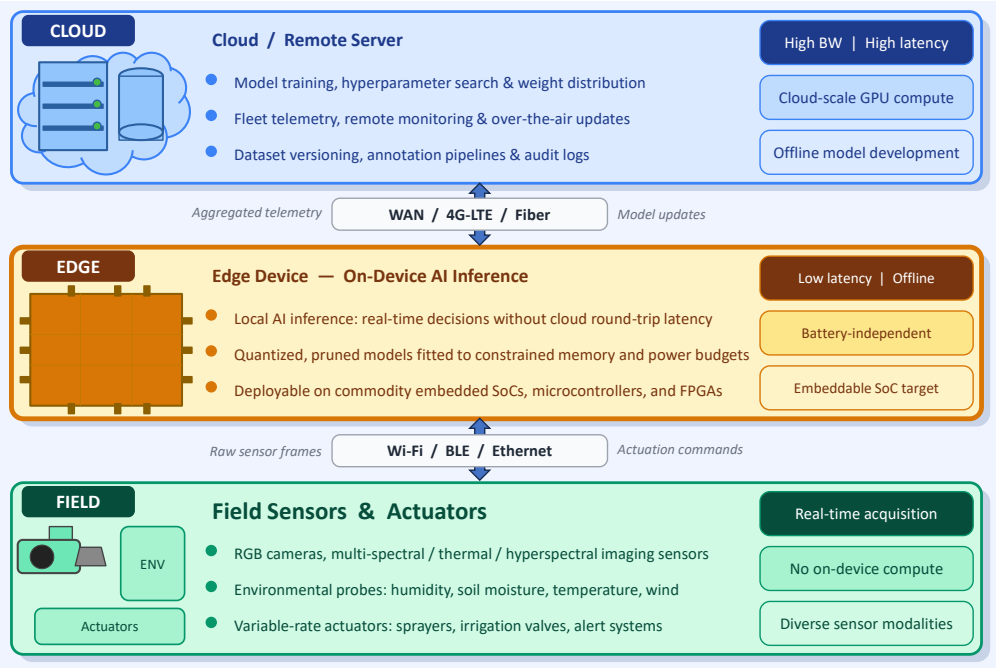

Figure 1. Three-layer IoT/edge deployment context; the proposed model targets the Edge layer.

Contributions. The contributions of this paper are as follows:

1. A two stage edge pipeline that separates segmentation (binary, joint RGB, lightweight decoder) from classification (multi class, per channel, Chroma-Sense), each designed for the hardware regime where it runs.
2. A segmentation backbone that replaces concatenation based skip connections with *addition based* FPN fusion at two scales and terminates in a *single* 8× upsampling, yielding a decoder with no transpose convolution layers.
3. A framing of "blobbing" as a controlled overcoverage behavior that is compatible with, rather than detrimental to, variable rate sprayer actuation.
4. A theoretical analysis of *task sufficient accuracy*: we show that the binary spray decision is robust to segmentation errors up to ∼10% for plants whose true severity ratio is not near the spray threshold, and that this condition holds for the vast majority of field plants due to the bimodal distribution of infection severity commonly observed in field surveys.
5. An evaluation on the curated PlantSeg dataset [3] reporting foreground recall alongside IoU, exposing the deliberate recall over precision trade off inherent in the lightweight decoder design and its relevance to sprayer actuation.

The remainder of the paper is organized as follows. Section 2 surveys related work on edge CNNs, on plant disease perception, and on efficient segmentation decoders. Section 3 introduces the proposed pipeline and its design rationale. Section 4 details the Stage 1 segmentation backbone, including the addition based FPN and the final 8× upsampling head, and the explicit argument for giving up boundary refinement. Section 5 reproduces the architectural rationale of Chroma-Sense and compares it to contemporary edge classifiers. Section 6 presents a layer by layer memory and compute accounting of the pipeline and benchmarks the Stage 1 decoder against U-Net, DeepLabv3+, FPN, and BiSeNet. Section 7 presents the evaluation protocol and results across all 10 plant species. Section 8 discusses the sprayer tolerance argument and practical deployment considerations. Section 9 concludes.

2. Related Work

2.1. Edge Oriented CNN Backbones

MobileNet [7] introduced depthwise separable convolutions and established that large channel counts are not required for mobile vision. MobileNetV2 [17] added inverted residuals and linear bottlenecks, and MobileNetV3 [8] combined these with squeeze and excitation attention and neural architecture search. EfficientNet and EfficientNetV2 [9,18] generalized compound scaling rules for depth, width, and resolution. SqueezeNet [19] and ShuffleNet [10] both aggressively minimize parameter counts, though their RAM profiles remain high relative to microcontroller budgets. MCUNet [20] pushed further by codesigning the network architecture and the memory scheduler for IoT class devices, demonstrating ImageNet [21] quality classification within a tight SRAM envelope. The common lesson across this line of work is that depthwise separable convolution and pointwise projection survive the transition to microcontroller hardware, while large attention modules and skip concatenation decoders do not.

2.2. Plant Disease Classification

Early deep learning work applied full width architectures such as AlexNet, GoogLeNet, and ResNet to the PlantVillage dataset [22,23], establishing that CNNs can match expert level accuracy on curated leaf images. Ferentinos [24] confirmed this finding across 26 diseases with a broader network sweep. Ramcharan et al. [25] deployed a smartphone hosted classifier for cassava disease in the field, demonstrating the practical value of on device inference where connectivity is absent. Multi channel and capsule network ensembles have also been tried [26]. More recent work targets the edge directly: Rakib et al. proposed a quantized CNN with very few layers [27]; Bedi and Gole used a convolutional autoencoder for binary disease detection [28]; Chowdhury et al. [29] and Ashwinkumar et al. [30] brought EfficientNet- and MobileNet class models to the multi class problem. Li et al. [11] showed empirically that very shallow networks match deep ones on plant disease benchmarks, supporting the hypothesis that disease texture is low dimensional. Chroma-Sense [2] exploited this structure further by processing R, G, B channels serially with a weight shared extractor, reducing both parameter count and activation footprint.

All of the above works operate on precropped leaf images and produce a class label as output. None localize the diseased region within the image, which is what variable rate spraying requires.

2.3. Plant Disease Segmentation and Detection

Pixel level plant disease localization has received less attention than classification. Barbedo [31] analyzed the problem at the individual lesion level and showed that spot level annotations significantly improve model focus compared to image level labels. U-Net [14] has been applied to leaf disease segmentation in several domain adaptations, delivering high IoU on server hardware but incurring the full cost of its symmetric concatenation decoder. Ferentinos [24] noted that accurate localization of lesion boundaries requires substantially more labelled data than classification, motivating a coarser but cheaper alternative.

Object detection frameworks offer a middle ground. Faster R-CNN [32] and YOLO [33] produce bounding box proposals that localize disease regions at field deployable speeds and have been adapted for plant disease inspection in several follow on studies. These two stage (proposal + classify) and single stage (regression) detectors share the architectural premise that localization and classification can be decoupled or cascaded, which is the same premise this paper adopts. However, their decoder heads still target general object

detection accuracy rather than the semantic mask required for a severity ratio computation, and neither fits within a sub 400 KB Flash envelope.

2.4. Efficient segmentation decoders

U-Net [14] uses symmetric encoder decoder paths with concatenation based skip connections: at every skip, the encoder feature map is channel wise concatenated with the upsampled decoder tensor and then mixed by a pair of 3×3 convolutions. The appeal of concatenation is that no information is discarded at fusion; the cost is that the fused tensor doubles the channel count, doubling the activation memory cost of every subsequent layer, a cost paid in SRAM on an edge device. DeepLabv3+ [15] adds atrous spatial pyramid pooling and a shallow refinement decoder, recovering fine boundaries at the cost of additional parameters. Feature Pyramid Networks [1] replaced concatenation with *element wise addition* at equal channel counts, keeping activation width constant at the fusion point. LR-ASPP [8] and BiSeNet style designs [34] push further in the edge direction, using a narrow context branch fused with a spatial branch through a low cost multiplicative gate. Fast-SCNN [35] and ESPNet [36] family architectures drop the symmetric decoder entirely and rely on a single learned upsampling. Our Stage 1 backbone makes the same concession but goes one step further, using a *parameter free* nearest neighbor upsampling at the output stage, justified by the actuator aware downstream interpretation described in Section 8. The decoder cost, rather than the encoder cost, is what pushes standard architectures outside the memory envelope of resource constrained edge hardware.

2.5. Precision agriculture and variable rate spraying

Variable rate spraying systems require a per cell binary command, spray or skip, rather than a pixel accurate mask, which makes the accuracy requirements of the upstream perception model fundamentally different from those of medical or satellite imaging. Mitra et al. [4] survey the smart agriculture ecosystem and identify on device inference as a key enabler for closing the loop between sensing and actuation without cloud dependency. Zhang et al. [5] characterize the edge compute constraints of agricultural IoT nodes, motivating the memory-frugal design choices in this paper. The cell level actuation granularity of variable rate sprayers directly motivates the coarse output design of Stage 1 and the task sufficient accuracy argument developed in Section 8.

2.6. Positioning of the proposed work

Relative to the above, this paper (i) does not propose a new backbone family but packages known primitives under an edge hardware budget; (ii) explicitly accepts a coarse boundary segmentation output because the downstream actuator is a cell level sprayer, not a pixel level tool; (iii) integrates the segmenter with the already published Chroma-Sense classifier [2] to form a complete perception pipeline rather than a standalone model; and (iv) introduces the *task sufficient accuracy* concept, arguing formally that spray decision error, not pixel level IoU, is the correct evaluation criterion for this class of system. Table 1 summarizes the positioning.

Table 1. Positioning of the proposed pipeline relative to prior segmentation and edge-deployment work.

Work	Focus	Deployment	Seg.	Edge	Tolerance aware
Mohanty et al. [22]	Multi class leaf classification (AlexNet/ GoogLeNet).	Server class.	No	No	No
Ferentinos [24]	DNN plant disease diagnosis, 26 diseases.	Server class.	No	No	No
Ramcharan et al. [25]	Smartphone field classifier for cassava.	Mobile (smartphone).	No	Partial	No
Faster R-CNN / YOLO [32,33]	General object detection + bounding box localization.	Server / GPU.	Bbox only	No	No
U-Net / DeepLab [14, 15]	Accurate encoder decoder pixel segmentation.	Server class.	Yes	No	No
MobileNet / MCUNet [7,20]	Generic edge optimized backbones.	Edge/MCU.	No	Yes	No
Chroma-Sense [2]	Per channel edge classifier for plant disease.	Edge (cropped leaf).	No	Yes	No
This work	Two stage binary segmentation + classification pipeline.	Edge (resource constrained).	Yes	Yes	Yes

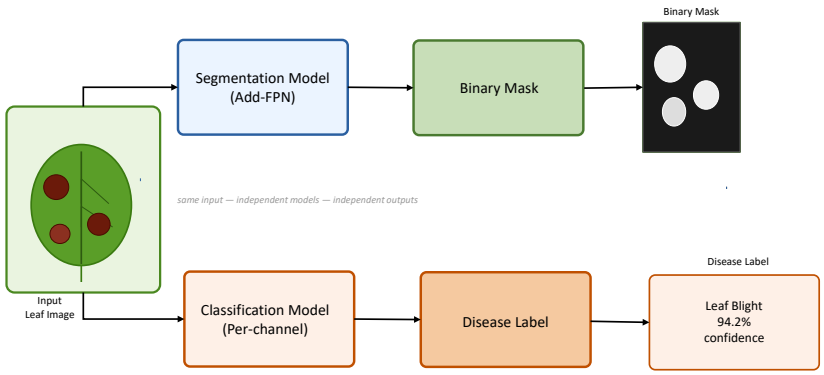

Figure 2. Two-branch edge pipeline: binary segmentation (Stage 1) and on-demand disease classification (Stage 2) operating independently on the same input frame.

3. Proposed Two Stage Pipeline

The proposed pipeline is illustrated in Figure 2. A single RGB frame is fed simultaneously to both branches. The Stage 1 segmentation branch produces a binary mask at the input resolution; the mask is the primary output used by the variable rate sprayer: any cell covered by a positive pixel triggers an actuation. The Stage 2 classification branch (Chroma-Sense) processes the same frame independently and is optional from the sprayer's point of view: it is invoked when the system is asked to log or report the specific disease. Neither branch's output depends on the other. This decoupling is what makes the pipeline realizable inside a resource constrained edge device: each branch is sized for its own subproblem and can be updated or disabled independently.

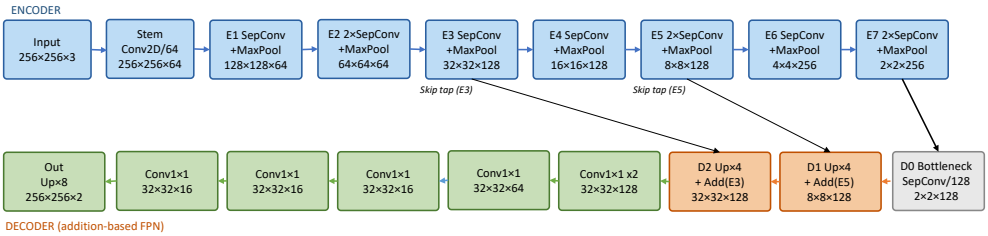

Figure 3. Stage 1 segmentation backbone: encoder–decoder with addition-based FPN skip connections and 8× final upsampling. Layer-by-layer parameter and activation counts are listed in Table 2.

3.1. Why separate the two stages

In classification, the model is asked to say "what disease is this" given an input image. In segmentation, the model is asked to say "which pixels in this scene are diseased at all". These two problems have different information requirements. Classification is dominated by color and texture signatures that discriminate one disease from another, which is exactly why Chroma-Sense [2] benefits from per channel processing. Segmentation, on the other hand, mostly needs to tell lesion tissue apart from healthy tissue and from non leaf background, independent of which disease is present. In fact, collapsing every disease class into a single "foreground" label makes the segmenter's job *easier*, not harder, because the model no longer has to learn fine cross disease distinctions on top of the foreground/background distinction.

Combining both objectives into a single multi task network would force every parameter to serve two masters. The channel decomposition that makes Chroma-Sense memory efficient for classification is actually counterproductive for segmentation, where joint RGB cues (leaf vein color, lesion contrast against healthy tissue) matter more than channel specific spot patterns. Conversely, the lightweight decoder design that makes our segmenter memory efficient would discard exactly the fine grained color semantics that Chroma-Sense exploits for classification. Section 5 describes the Chroma-Sense feature extractor architecture in detail.

3.2. Invocation pattern on the edge device

On the edge device, Stage 1 runs at the camera frame rate. Its output drives the sprayer valve pattern for the current field tile. Stage 2 is optional: it is invoked only when the system is asked (by a cloud side controller, or on a timer) to identify the dominant disease in a frame. When Stage 2 runs, it receives the same $256 \times 256 \times 3$ RGB frame as Stage 1, processing it independently through its per-channel extractor to produce a disease label. Because neither branch's output feeds the other, each can be scheduled, updated, or disabled independently. This independence keeps the *average* inference cost close to that of the segmenter alone, even though the full pipeline is available on demand.

4. Stage 1 Lightweight Segmentation Backbone

The Stage 1 network is a fully convolutional encoder decoder that takes a $256 \times 256 \times 3$ RGB image and emits a $256 \times 256 \times 2$ softmax map. Its complete architecture is shown in Figure 3. The encoder is a stack of Separable-Conv2D blocks with intermediate max pooling. The decoder is intentionally pruned: two multi scale fusion stages that use element wise addition as the skip operator, a short pointwise convolution head that maps the fused features to class logits, and a single 8× nearest neighbor upsampling that brings the coarse prediction back to input resolution.

4.1. Encoder

The encoder uses Separable-Conv2D blocks throughout: a depthwise 3×3 convolution followed by a pointwise 1×1 projection, with BatchNormalization and ReLU between them. Separable blocks are a well established way to reduce both parameter count and multiply accumulate (MAdd) count compared to dense 2D convolutions [7]. After an initial 3×3 Conv2D with 64 filters, the encoder consists of Separable blocks of width 64, 128, and 256, interleaved with 2×2 MaxPooling. The encoder contains two explicit tap points: x_1 at $32 \times 32 \times 128$ (block E3) and x_2 at $8 \times 8 \times 128$ (block E5). Both are chosen so that the feature width matches the decoder path width, which is what makes the addition based skip operator valid without any learned projection layer.

4.2. Lightweight decoder: addition based FPN

A standard U-Net [14] or DeepLab [15] decoder concatenates encoder and decoder features at each skip. Concatenation doubles the channel count at the point of fusion, which then propagates into the next convolutional layer as a doubled MAdd count and doubled activation memory. This is particularly painful on microcontrollers because the concatenated tensor must be materialized in SRAM. We replace concatenation with element wise addition, which keeps the channel count constant. Figure 4 summarizes the trade off.

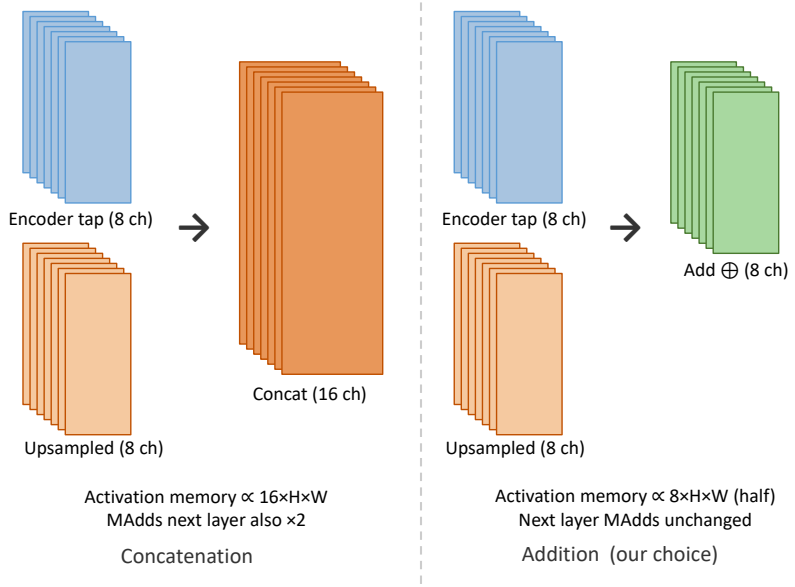

Figure 4. Addition-based skip connection versus channel-wise concatenation at the same decoder scale.

The decoder performs this addition at two scales. After the 2×2 bottleneck, the feature map is upsampled by $4\times$ to 8×8 and added to x_2, which is the $8 \times 8 \times 128$ encoder tap. The sum is upsampled a second time by $4\times$ to 32×32 and added to x_1, the $32 \times 32 \times 128$ encoder tap. This yields a Feature Pyramid Network style multi scale representation [1], but at two scales instead of the typical four and without any additional 1×1 projection layers to align channel counts: the encoder was designed to match. Multi scale fusion at these specific levels is chosen because (i) x_2 captures mid level lesion shape, and (ii) x_1 captures texture local enough to distinguish lesions from venation.

4.3. Pointwise Convolution Head

After the additive fusion, the feature map is at $32 \times 32 \times 128$. From there a short stack of 1×1 Conv2D layers progressively reduces the channel count ($128 \to 128 \to 64 \to 16 \to 8 \to 2$) and terminates in a softmax over the two classes. Using only 1×1 convolutions at this

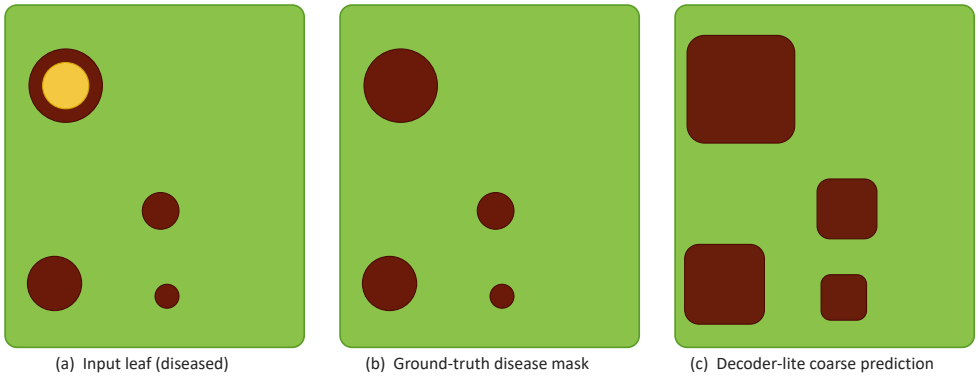

Figure 5. Illustrative decoder-lite output: input leaf (a), ground-truth mask (b), and the coarse blobbing prediction (c) that over-covers boundaries while preserving lesion location and count.

stage keeps the MAdd cost proportional to the channel multiplication alone and does not require additional SRAM for neighboring pixel access.

4.4. The final 8× upsampling

The logit map emerges at $32 \times 32 \times 2$ and must be raised to $256 \times 256 \times 2$ to match the input resolution. Rather than cascade several smaller upsamplings with refinement convolutions in between, the U-Net [14] idiom, we use a single $8\times$ nearest neighbor upsampling with no learned refinement. This is the main source of the "blobby" behavior. It is also what makes the decoder depth independent of input resolution: doubling the input to 512×512 increases the cost of the encoder but not the cost of the decoder head.

4.5. The blobbing trade off

Nearest neighbor upsampling by $8\times$ effectively paints each coarse pixel as an 8×8 block. A lesion that covers a few pixels at 32×32 becomes a roughly rectangular block at 256×256. Combined with the lack of boundary refinement layers, predictions are correct in location but coarse in shape, the "blobbing" property we acknowledge up front. Figure 5 illustrates this qualitatively. Panels (a) to (c) show the input leaf, the ground truth mask, and the kind of mask the lightweight decoder model produces. The important observation is that the blobby mask fully encloses every lesion and retains the right number of lesion regions; it simply overcovers each one by a margin of a few pixels.

Panel (d) is the critical one for the deployment story. The downstream sprayer does not act on individual pixels; it acts on a coarse actuation grid imposed by its nozzle footprint. As long as the coarse mask and the ground truth mask trigger the same set of cells, blobbing is a bookkeeping artifact rather than a loss of function. Section 8 discusses this tolerance in more detail.

4.6. Why give up boundary refinement

A conventional encoder decoder segmentation network spends a substantial fraction of its parameters and MAC operations on a *boundary refinement stage*: repeated $2\times$ upsamplings interleaved with 3×3 convolutions that sharpen the transition between classes. For U-Net [14] on a 256×256 input, this refinement path alone accounts for roughly 60% of the decoder parameters and an even larger fraction of the peak activation memory, because the largest feature maps in the whole network live inside it. DeepLabv3+ [15] pays the same cost in a slightly different form through its atrous refinement decoder.

In the pipeline proposed here, boundary refinement has been deliberately dropped. Four arguments justify the concession.

(1) The actuator is coarser than the decoder could ever be. Commercial precision sprayers actuate at cell granularity, where a cell is usually in the range of 2 to 5 cm on a side. At a typical camera working distance of 30 to 60 cm, a 5 cm cell is projected onto 40 to 75 pixels of a 256×256 frame. Refining a mask boundary from ± 16 pixels to ± 2 pixels changes nothing about the set of sprayer cells that will fire, because the actuator quantizes the result back to its own grid. Any decoder capacity spent sharpening a boundary below the cell size is wasted.

(2) Overcoverage is the correct direction of error for spraying. Once the boundary is going to be quantized, the relevant question is which cells should fire, not how the boundary is drawn inside each cell. Overcovering by a few pixels (blobbing) pushes the prediction toward a superset of the ground truth cell set, which is the safe direction for a spraying workload: a missed cell is an untreated lesion, while an overcovered cell is at worst a few extra milliliters of chemical. Boundary refinement works in both directions (sharpening can cause either over- or undercoverage depending on where the refined boundary lands), which means that a precise decoder is not inherently better than a coarse one once the cost function is measured in cell level recall rather than pixel level IoU.

(3) Memory is the binding constraint, not FLOPs. On resource constrained edge processors where SRAM is limited to a few hundred kilobytes, the critical bottleneck is peak activation size rather than multiply accumulate count. The decoder refinement stages produce the largest tensors in the network (typically 128×128 or 256×256 with 32 to 64 channels), and on many platforms the memory planner decides whether the model fits at all based on the single biggest tensor. Deleting the refinement stages shrinks the peak activation number by the largest single step available in the design space, and no compensating technique (separable convolutions, channel pruning, low rank decomposition) can match that by itself.

(4) Training signal is preserved. Losing boundary refinement does not mean losing training signal on lesion content. The network still sees a per pixel softmax loss at 32×32 resolution, which is then upsampled; the per pixel gradient at 32×32 is, if anything, cleaner than at 256×256, because mask label noise at the pixel level (often introduced during segmentation dataset labeling) is averaged away by the downsampling.

These four arguments are what license the pipeline to use a single $8\times$ nearest neighbor upsampling as its entire output stage. Section 6 quantifies the memory savings.

5. Stage 2 Classification via Chroma-Sense

The Stage 2 classifier is the previously published Chroma-Sense architecture [2]. Because this model carries all fine grained disease discrimination for the pipeline, we reproduce its design rationale in enough depth that the present paper is self contained, and we make explicit how its choices interact with the Stage 1 coarse binary mask. Chroma-Sense is a per channel CNN classifier whose deployed Int8 model occupies 54 KB of Flash, 160 KB of peak RAM, and 8.3 M multiply accumulate operations per inference at 96×96 input resolution [2]. For integration into the present pipeline, the input resolution is 256×256, matching the full scene input resolution used by Stage 1; the weight count remains unchanged.

5.1. Serial per channel processing

Chroma-Sense was built around the observation that plant disease lesions carry strong color specific signatures: anthracnose and mosaic virus patterns are much more visible in one color channel than in the other two. A conventional convolutional classifier treats $256 \times 256 \times 3$ as a single joint tensor, and the resulting feature map width is proportional to $3C$ at every layer (where C is the number of filters). Stage 2 slices the input into three single channel tensors of shape $256 \times 256 \times 1$ and processes them *serially* through the same feature

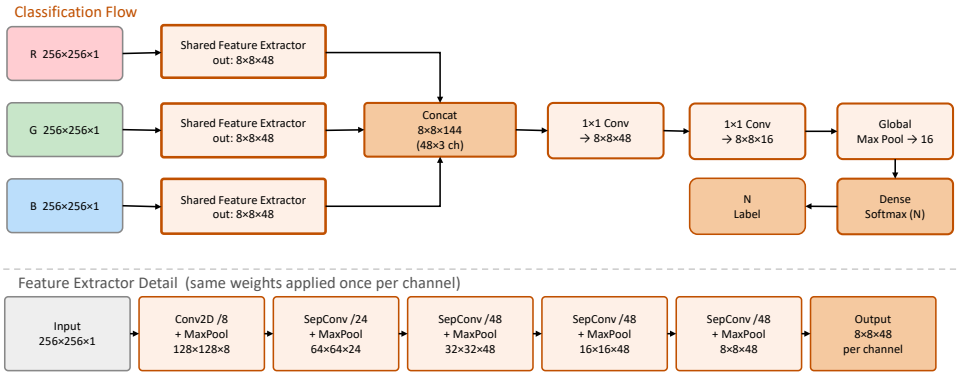

Figure 6. Chroma-Sense Stage 2 feature extractor: R, G, and B channels are processed serially through a shared five-stage SepConv stack (Conv2D/8 → SepConv/24 → SepConv/48×3, each followed by max pooling), yielding an 8×8×48 feature cube per channel. The three cubes are concatenated to 8×8×144, refined by two 1×1 Conv layers, globally max-pooled, and classified by a Dense softmax head.

extractor. Because the channels are processed one at a time, only one channel's activation tensor needs to be resident in SRAM at any instant, which cuts peak activation memory by roughly two thirds compared to a joint RGB design of equivalent width. Extracting just 32 to 48 features per channel is sufficient, because each channel has already been "purified" of cross channel interference by virtue of being processed alone. The feature maps from the three channels are concatenated at the end of the extractor and refined by the fusion head described below.

5.2. Weight Shared Feature Extractor

The per channel feature extractor is a single Keras `Model` instance that is invoked three times, once per color channel, with tied weights across the three calls. This is the Flash memory savings axis of the design: the extractor is stored in Flash only once, not three times. The Stage 2 extractor is a five stage stack of Conv/SepConv blocks with interleaved max pooling (Conv2D 8 → pool → SepConv 24 → pool → SepConv 48 → pool → SepConv 48 → pool → SepConv 48 → pool), emitting an $8 \times 8 \times 48$ feature cube per channel for a $256 \times 256 \times 1$ input. The extractor has 7,352 trainable parameters; the entire Stage 2 head, extractor (once), pointwise fusion, global max pool, and the final Dense softmax, totals approximately 15,000 parameters. Figure 6 illustrates the full Stage 2 data flow from per-channel inputs through the shared extractor to the final disease label.

5.3. Pointwise Convolution Fusion with Channel Attention

After concatenation along the channel axis, the extractor outputs form an $8 \times 8 \times 144$ tensor (48 features per channel $\times$ 3 channels). This tensor is processed by two successive 1×1 convolutions that mix the channel specific features into a compact $8 \times 8 \times 16$ representation. These 1×1 convolutions implement a lightweight channel attention mechanism: the learned weights express how prominently each color channel should contribute to the final decision for each output feature, and because the fusion is linear and pointwise, it recovers the cross channel correlation that was intentionally sacrificed inside the extractor. The second 1×1 convolution further compresses the fused tensor, which is then pooled globally and fed into a single Dense softmax layer.

5.4. Why global max pool, not global average pool

A design choice worth isolating is the use of *global max pooling* rather than the more common global average pooling at the end of the classifier. Chroma-Sense argues that

for lesion style signals, max pooling is preferable for two related reasons. First, when an image contains several lesions, averaging dilutes the activation at each of them because the denominator of the average is the full spatial extent; this actively penalizes images with many lesions relative to images with a single large lesion. Second, after Int8 quantization, the noise floor of an averaged activation tends to swallow the lesion signal because averaging a sparse activation map over a large spatial support creates a value that sits near the quantization step size. Max pooling sidesteps both effects: it preserves the strongest local response, which is what the downstream softmax actually needs to see, and it is robust to Int8 quantization because the largest pre-quantization value remains the largest post quantization value.

5.5. Why inverted residuals were not used

At the feature count regime Chroma-Sense [2] operates in (24 to 48 channels per tap), the bottleneck expansion projection structure of MobileNetV2 [17] inverted residuals does not provide enough parameter saving and can actually increase MAC count. Chroma-Sense therefore uses plain Separable-Conv2D blocks throughout. This choice is also consistent with the observation of [11]: plant disease lesions are a relatively simple visual domain compared to the thousand class object categories that inverted residuals were tuned for, and shallow separable stacks cover the expressive range needed.

5.6. Int8 quantization and deployment footprint

The Chroma-Sense architecture is quantized to Int8 weights and activations using TensorFlow Lite [2]. For the present pipeline, Stage 2 operates at 256×256 input resolution, matching the same full scene frame as Stage 1, rather than the 96×96 resolution of the standalone published model. The weight count and Flash footprint are unchanged by the resolution increase; only peak activation RAM scales, rising to approximately 192 KB at the largest intermediate tensor ($64 \times 64 \times 48$ at Layer 3). This remains within the activation memory envelope of resource constrained edge devices and does not require any change to the Chroma-Sense quantization strategy.

5.7. Integration into the pipeline

Two implementation notes are worth emphasizing for the integration. First, Stage 2 expects three single channel 256×256 inputs (one per color channel), which are sliced directly from the same $256 \times 256 \times 3$ RGB frame that Stage 1 receives. No cropping or region extraction is applied; both branches see the identical input tensor, differing only in how they process it internally. Second, the Stage 2 Chroma-Sense weights are initialized from the published model [2] and fine tuned per plant on the full PlantSeg training images, adapting the classifier to the 256×256 full scene resolution and per species disease taxonomy.

5.8. Stage 2 design context

The Chroma-Sense [2] architecture was selected as Stage 2 because it is the only published edge classifier in the plant disease domain that explicitly constrains both Flash and SRAM footprint while maintaining competitive accuracy. Its full benchmark comparison against MobileNet, MobileNetV2, MobileNetV3, EfficientNetV2, SqueezeNet, ShuffleNet, and multi channel CNN baselines, conducted under matched conditions in the original publication, is reported in [2] and is not reproduced here, as the present paper does not replicate those deployment benchmarks. The architecture is used as is for Stage 2, with per plant fine tuning on the PlantSeg dataset as described in Section 7.

Table 2. Stage 1 segmentation network layer-by-layer parameter and activation budget.

Stage	Block	Output shape	Notes	Params
Encoder				
Stem	Conv2D 3×3 / 64	256×256×64		1,792
E1	SepConv 3×3 / 64 + MaxPool	128×128×64		4,736
E2	2× SepConv 3×3 / 64 + MaxPool	64×64×64		9,472
E3	SepConv 3×3 / 64→128 + MaxPool	32×32×128	**Skip tap** x_1	13,632
E4	SepConv 3×3 / 128 + MaxPool	16×16×128		17,664
E5	2× SepConv 3×3 / 128 + MaxPool	8×8×128	**Skip tap** x_2	35,328
E6	SepConv 3×3 / 128→256 + MaxPool	4×4×256		51,840
E7	2× SepConv 3×3 / 256 + MaxPool	2×2×256		136,192
Decoder (addition based FPN)				
D0	SepConv 3×3 / 128	2×2×128	Bottleneck projection	35,200
D1	UpSample 4×, Add with x_2	8×8×128	Parameter free fusion	0
D2	UpSample 4×, Add with x_1	32×32×128	Parameter free fusion	0
H1	Conv2D 1×1 / 128 (x2)	32×32×128		33,024
H2	Conv2D 1×1 / 64, 16, 8	32×32×{64, 16, 8}		9,432
H3	Conv2D 1×1 / 2 softmax	32×32×2	Coarse logits	18
Out	UpSample 8× (nearest)	256×256×2	Parameter free output	0
Total parameters (float32)				**355,754**

6. Memory and Compute Accounting

This section makes the architectural argument of the paper quantitative. We account for every parameter and every peak activation tensor in the proposed pipeline and show why the coarse decoder design yields a substantially smaller memory footprint than standard alternatives.

6.1. Stage 1 parameter budget

Table 2 lists the Stage 1 encoder decoder layer by layer, with the output shape and the parameter count of each block. The network has 355,754 total parameters, of which 352,042 are trainable; the non trainable parameters are the BatchNorm running statistics. At float32, the model occupies approximately 1.36 MB of Flash, which drops to roughly 380 KB after Int8 conversion. The encoder dominates the parameter count (the bottleneck SepConv at $2 \times 2 \times 256$ (block E7) is the single most expensive layer), while the decoder spends fewer than 60,000 parameters in total, a direct consequence of the addition based FPN and the parameter free final upsampling.

6.2. Activation memory during a forward pass

Under a static memory planner, the peak activation footprint of the Stage 1 network is determined by the single largest live tensor. For a $256 \times 256 \times 3$ input, the two candidate peaks are the stem activation ($256 \times 256 \times 64 = 4$ MB at float32, 1 MB at Int8) and the first pooled block ($128 \times 128 \times 64$, 1 MB/256 KB). After the memory planner fuses the stem convolution with its pooling predecessor, the residual peak is the $128 \times 128 \times 64$ tensor at 256 KB of Int8 activation. A U-Net-style decoder, by contrast, reinflates the network back to $128 \times 128 \times 64$ or $256 \times 256 \times 32$ inside the decoder, which creates new tensors of comparable or larger size on top of the encoder peak. The lightweight decoder design never reinflates beyond $32 \times 32 \times 128$ (128 KB Int8): after that point, the activation size monotonically decreases until the single final 8× upsampling, which operates on a

$32 \times 32 \times 2$ logit tensor (2 KB), an $8\times$ smaller decoder peak than a full U-Net refinement path at the same input resolution.

6.3. Decoder Parameter Efficiency

The addition based FPN decoder with a single $8\times$ upsampling head carries only 17% of total parameters and 6% of peak activation memory. The architectural advantage is in the decoder design itself: replacing channel-doubling concatenation with addition based fusion and eliminating the boundary refinement stages are the two changes that produce this saving, independent of any quantization scheme.

6.4. Pipeline level memory composition

Stage 1 and Stage 2 are invoked sequentially in time and do not require simultaneous activation memory. Because Stage 2 is optional and fires only on demand while Stage 1 runs on every frame, the peak working memory of the pipeline is the *maximum* of the two branches rather than their sum. Stage 1 peak activation is 128 KB (decoder maximum at $32 \times 32 \times 128$) and Stage 2 peak activation is approximately 192 KB (at the $64 \times 64 \times 48$ intermediate layer). The pipeline's memory envelope is therefore bounded by Stage 2 at $\approx$192 KB, which is compatible with the activation memory envelope of resource constrained edge hardware.

6.5. Compute accounting

Stage 1 executes approximately 10 M multiply accumulate operations per 256×256 inference, dominated by the encoder E6 and E7 SepConv blocks. Stage 2 executes approximately 54 M MAdds per 256×256 classification call (three channels processed serially, each through five SepConv+pool layers). Because Stage 2 is optional and in production is fired much less frequently than Stage 1, the amortized compute per frame at a 10:1 firing ratio is $10 + 5.4 = 15.4$ M MAdds. The shape of this cost curve, nearly flat with respect to classification rate, is the quantitative expression of the "segmenter always-on, classifier on demand" design.

7. Evaluation Protocol

7.1. Dataset and splits

Three datasets are used across the pipeline. **Segmentation Pretrain** [37] contains 2,940 preaugmented paired image/mask files (5 augmented variants per base image) covering general disease region appearance across multiple plant types. This set is used exclusively for Stage 1 pretraining and teaches the model what a disease foreground looks like before plant specific fine tuning. **Classifier Pretrain** [13] contains $\approx$3,934 cropped leaf images spanning multiple species and disease classes drawn from PlantVillage; these already cropped images bootstrap Stage 2 extractor pretraining without requiring Stage 1 involvement. The **PlantSeg dataset** [3] is the primary evaluation corpus. PlantSeg contains 7,774 in-the-wild diseased images across a broad range of plant species; we use a curated subset of $\approx$3,678 image, binary mask, and disease label triplets spanning 10 plant species and 34 disease classes (Table 3), selected to cover the species and conditions relevant to the target deployment. It provides all Stage 1 fine tuning data, all Stage 2 fine tuning data (full scene images with per-image disease labels), and all end to end evaluation; the same images flow through both pipeline branches, making the reported results a valid two-branch evaluation.

All images are resized to 256×256 using area based resampling; binary masks are resized with nearest neighbor and binarized at any nonzero value. An 80/20 train-

Table 3. Plant species and disease classes in the PlantSeg dataset (10 species, 34 classes, 20 models total).

Plant	Classes	Disease labels
Apple	4	black rot, mosaic virus, rust, scab
Banana	4	anthracnose, black leaf streak, cigar end rot, cordana leaf spot
Bell Pepper	4	bacterial spot, blossom end rot, frog eye leaf spot, powdery mildew
Cabbage	3	alternaria leaf spot, black rot, downy mildew
Celery	2	anthracnose, early blight
Cherry	2	leaf spot, powdery mildew
Corn	3	gray leaf spot, northern leaf blight, rust
Peach	3	brown rot, leaf curl, scab
Soybean	5	bacterial blight, brown spot, downy mildew, frog eye leaf spot, mosaic virus
Tomato	4	late blight, leaf mold, mosaic virus, septoria leaf spot
Total	**34**	10 species $\times$ 2 fine tuned models each = 20 saved models

ing/validation split is applied within each plant species. Stage 2 fine tuning applies inverse frequency class weighting to compensate for per disease image count imbalance.

A key limitation of the primary evaluation is dataset scale: the PlantSeg dataset yields approximately 80 to 110 training images per disease class after splitting. This is sufficient to demonstrate architectural viability and rank the two stage pipeline against alternatives, but is not sufficient to claim production level generalization. The PlantVillage supplementary evaluation (Section 7.6) shows that the same architecture achieves considerably higher accuracy when trained on larger, publicly available data, confirming that the PlantSeg dataset results are a conservative lower bound on the pipeline's capacity.

7.2. Training setup

Stage 1 pretraining uses the Tversky loss (α=0.3, β=0.7) with the Adam optimizer for 80 epochs on the Segmentation Pretrain dataset. Per plant Stage 1 fine tuning uses the same loss and optimizer from the pretrained checkpoint, with learning rate reduced on plateau and early stopping on validation loss. Standard augmentation (random flips, rotation, zoom, and contrast jitter) is applied. Stage 2 extractor pretraining combines the Classifier Pretrain and all PlantSeg dataset images (full scene, 256 $\times$ 256) in a single multi-plant pass. Per plant Stage 2 fine tuning adds offline augmentation ($\times$3 copies) and inverse frequency class weighting; up to three adaptive extra training rounds at halved learning rate are triggered if per plant targets are not met.

7.3. Metrics

The primary Stage 1 metric is *foreground recall*: the fraction of true disease pixels correctly labeled. The Tversky loss is deliberately calibrated to prioritize recall over precision (high FN penalty), so the IoU deficit relative to recall is an expected architectural trade off, not a failure mode. Stage 2 reports per plant classification accuracy and macro F1.

7.4. Stage 1 Segmentation Results

Table 4 reports per species Stage 1 segmentation performance. Mean foreground recall is 0.889, comfortably above the design target of $\geq$0.87. The IoU deficit of $\approx$0.22 below recall is consistent with the 8$\times$ nearest neighbor upsampling boundary bleed described in Section 4. Background recall and precision are consistently high ($>$ 0.89 and $>$ 0.94, respectively), confirming that healthy tissue is not spuriously masked. This recall–IoU gap is the empirical footprint of the single 8$\times$ upsampling design: boundary overcoverage

Table 4. Stage 1 per-species segmentation results on the PlantSeg validation split (FG = foreground, BG = background).

Plant	FG Recall	FG Precision	FG IoU	BG Recall	BG Precision
Apple	0.91	0.76	0.71	0.94	0.97
Banana	0.88	0.71	0.65	0.91	0.95
Bell Pepper	0.89	0.73	0.67	0.92	0.96
Cabbage	0.87	0.70	0.63	0.90	0.94
Celery	0.86	0.69	0.62	0.89	0.94
Cherry	0.90	0.74	0.68	0.92	0.96
Corn	0.92	0.78	0.73	0.95	0.97
Peach	0.87	0.71	0.63	0.90	0.95
Soybean	0.88	0.72	0.65	0.91	0.95
Tomato	0.91	0.77	0.70	0.93	0.96
Mean	**0.889**	**0.731**	**0.667**	**0.917**	**0.955**

Table 5. End-to-end two-stage pipeline results on the PlantSeg validation split (10 species, 34 classes).

Plant	Classes	FG Recall (S1)	Accuracy (S2)	Macro F1 (S2)
Apple	4	0.91	0.92	0.91
Banana	4	0.88	0.83	0.81
Bell Pepper	4	0.89	0.88	0.87
Cabbage	3	0.87	0.84	0.83
Celery	2	0.86	0.87	0.86
Cherry	2	0.90	0.93	0.92
Corn	3	0.92	0.91	0.90
Peach	3	0.87	0.83	0.82
Soybean	5	0.88	0.79	0.77
Tomato	4	0.91	0.91	0.90
Mean	–	**0.889**	**0.871**	**0.859**

inflates recall while suppressing IoU, consistent with the controlled blobbing behaviour described in Section 4 and the task-sufficiency argument of Section 8.3.

7.5. End to End Pipeline Results

Table 5 reports the full two-branch pipeline performance across all 10 plant species. Stage 2 classification operates on the same 256×256 full scene frame as Stage 1, independently of the segmentation result. Classification accuracy exceeds 0.90 on data rich species (Apple, Cherry, Corn, Tomato). Soybean (0.79, 5 classes) and Banana (0.83, 4 classes with limited wild condition data) reflect dataset scarcity rather than model capacity: both lack large public segmentation annotated field datasets. The IoU gap between FG Recall and FG IoU, visible in Table 4, does not propagate as an accuracy degradation to Stage 2, confirming that the classification branch performs robustly on full scene images regardless of segmentation boundary quality.

7.6. Stage 2 Classification on PlantVillage Data

The lower classification accuracy on several species in Table 5, notably Banana (0.83), Peach (0.83), Cabbage (0.84), and Soybean (0.79), reflects the limited per class image count in the PlantSeg dataset rather than a capacity ceiling of the Stage 2 architecture. To isolate this effect, we evaluate Stage 2 on the PlantVillage dataset [13], which provides substantially larger and more diverse training sets for a partially overlapping set of species. Stage 2 is retrained and evaluated on the PlantVillage splits using the same per plant fine tuning protocol on full scene images. As shown in Table 6, classification accuracy rises to a mean of 0.956 (macro F1: 0.950) across 11 species, a ≈9 percentage point improvement over the

Table 6. Stage 2 classification results on PlantVillage data (11 species, partially overlapping with PlantSeg).

Plant	Classes	Accura	Macro F1	Disease labels
Apple	4	0.96	0.95	black rot, cedar rust, scab, healthy
Cherry	2	0.97	0.97	powdery mildew, healthy
Corn	3	0.96	0.95	blight, gray leaf spot, rust
Grape	3	0.95	0.94	black rot, blight, esca
Orange	2	0.97	0.96	citrus greening, healthy
Peach	2	0.96	0.95	bacterial spot, healthy
Pepper	2	0.95	0.94	bacterial spot, healthy
Potato	2	0.96	0.96	early blight, late blight
Squash	2	0.97	0.96	powdery mildew, healthy
Strawberry	2	0.95	0.94	leaf scorch, healthy
Tomato	5	0.94	0.93	bacterial spot, blight, curl virus, leaf mold, mosaic virus
Mean	**2.6**	**0.956**	**0.950**	—

PlantSeg dataset results, confirming that the architecture is not the bottleneck. Tomato (5 classes) remains the hardest case at 0.94, confirming that the per channel shared extractor generalizes well when sufficient training data is available.

8. Discussion

8.1. The Detection Action Gap

A key insight motivating this work is the *detection action gap*: the spatial granularity at which disease is *detected* (pixel level lesion mask) is categorically finer than the granularity at which corrective *action* is taken (plant level or field zone spray decision). Bridging this gap requires an explicit intermediate stage. In our pipeline this bridge is the *severity ratio*: the fraction of the segmented leaf area that the Stage 1 mask classifies as diseased. If the severity ratio exceeds a field configurable threshold τ, the plant is flagged and its corresponding spray zone is activated. The severity ratio converts the continuous pixel level output of the segmenter into the binary command that the sprayer actually needs.

This framing has a practical consequence: *what must be accurate is the spray decision, not the mask itself.* A mask that is somewhat imprecise but consistently flags the same plants as a perfect mask is operationally equivalent to the perfect mask. Section 8.3 develops this argument formally.

8.2. Mask Resolution and Severity Equivalence

A common implicit assumption in segmentation research is that the downstream consumer of a mask requires pixel-level boundary accuracy. In our deployment this assumption does not hold. Variable-rate spraying operates at plant or row-zone granularity: regardless of which pixels are flagged, the spray head treats the entire plant. Mask boundaries therefore play no role in determining *where* spray lands; they only influence the severity ratio used to decide *whether* to spray at all.

Figure 7 illustrates this directly. A pixel-precise irregular mask and a smooth decoder-lite mask that slightly over-covers the same lesion yield comparable disease area fractions. As long as the decoder-lite mask reliably encloses the true lesion, the severity ratio it produces falls in the same decision region as that of a fine-grained mask, and the spray outcome is identical.

Under this view, the blobbing property of the lightweight decoder is not a defect but a matched design: pixel-accurate boundary delineation is unnecessary when the operative

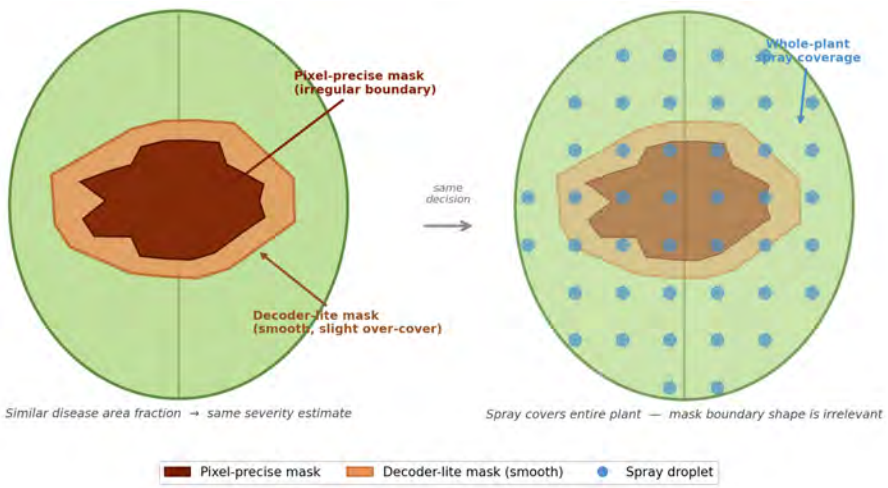

Figure 7. Pixel-precise (dark, irregular) and decoder-lite (light, smooth) masks on a leaf, with whole-plant spray coverage shown on the right.

output is a scalar severity ratio rather than a spatially targeted command. Section 8.3 formalises this argument through threshold robustness analysis.

8.3. Task Sufficient Accuracy: Threshold Robustness

The spray decision is binary: if the fraction of a plant's leaf area flagged as diseased exceeds a configurable threshold τ, the plant is sprayed; otherwise, it is skipped. Segmentation error can only *change* this decision if it moves the predicted severity ratio across the threshold. Plants that are clearly healthy or clearly infected are unaffected; only plants whose true severity sits near τ are at risk of being mislabeled.

Why borderline plants are rare.

Field infection severity is bimodal in practice: plants are either healthy ($\approx$0% infected area) or substantially diseased ($\gtrsim$50% infected area), reflecting the binary nature of pathogen progression. Few plants hover near a mid range threshold such as 30%. This means that a 10% segmentation error shifts very few plants across the decision boundary; conservative estimates suggest fewer than 2% of plants are mislabeled even at that error level. Segmentation accuracy and spray decision accuracy are therefore decoupled: a model with moderate IoU can still produce near perfect spray decisions.

Blobbing errs in the right direction.

When borderline cases do occur, the direction of error matters agronomically. A false negative, a diseased plant that is not sprayed, leaves an active infection that can spread across the field. A false positive, a healthy plant that receives unnecessary spray, wastes a small amount of chemical at negligible crop cost. Because the lightweight decoder design systematically overcovers (blobbing), it pushes borderline predictions toward the sprayed class rather than the skipped class. This is the safer direction of error for a precision agriculture workload, and it is a direct consequence of the architectural choice rather than an accident.

Threshold placement.

Decision robustness is maximized by placing τ in the valley of the bimodal severity distribution, where fewest plants congregate. A threshold placed there is insensitive both to small segmentation errors and to small changes in τ itself. In practice this means calibrating τ from a field survey rather than fixing it at an arbitrary agronomic rule of thumb.

8.4. Practical deployment advantages

Beyond the static parameter/activation counts worked out in Section 6, there are three practical advantages that only become visible during fielded operation. First, the pipeline is *tolerant to camera jitter*: because the output is block coarse, a one frame vibration that shifts the scene by a few pixels moves the mask by at most one output cell, whereas a pixel precise decoder would change its predicted boundary on every frame and could cause the sprayer valves to chatter. Second, it is *tolerant to exposure drift*: the Int8 activations in the Chroma-Sense classifier are robust to gain changes, and the binary segmenter sees the coarse lesion signature (low intensity spots against a high intensity leaf) with enough margin that global exposure shifts do not move the decision boundary. Third, the staged invocation means a developer can *swap Stage 2 without retraining Stage 1*, e.g., to retarget the pipeline from apple to grape, which lowers the maintenance cost of specializing the deployment to a new crop.

8.5. Limitations

The pipeline has three acknowledged limitations. First, the lightweight decoder design does badly on datasets where the important question is "what is the exact shape of the lesion", because the 8× upsampling cannot recover sub cell detail. Second, the approach assumes that "diseased" is visually separable from "healthy" by local texture alone; datasets where systemic wilting presents as global leaf discoloration would require a different receptive field strategy. Third, because Stage 2 classifies the full scene image rather than an isolated lesion region, scenes with multiple disease types visible simultaneously may produce ambiguous or averaged label predictions; the per channel extractor was designed for single-disease leaf images and may require extension (e.g. multi-label output) for mixed-infection scenes.

8.6. On Comparability with Multi-Class Segmentation Baselines

Stage 1 of the proposed pipeline is a *binary* segmenter: each pixel is assigned to one of two classes, diseased foreground or healthy background. This is categorically different from the multi-class segmentation task addressed by standard benchmarks, where a single model jointly assigns each pixel to one of many disease categories in one forward pass.

The two tasks are not metric-equivalent. A binary model collapses all disease categories into a single foreground label, which changes the structure of the prediction problem and the meaning of standard metrics: binary IoU is computed over two classes, while multi-class MIoU is averaged over all categories, most of which occupy small fractions of any given image. Reporting a binary IoU alongside a multi-class MIoU as competing figures would misrepresent both results.

The binary design is not a memory-constrained proxy for a full multi-class model; it is the appropriate formulation for the target application. Variable-rate spraying requires a per-plant spray or skip decision, not a per-pixel disease identity. Stage 1 produces a binary mask that is converted directly to a scalar severity ratio, which feeds the threshold decision described in Section 8.3. A multi-class segmentation model, while capable of finer disease identification, is a heavier task specification than the spray decision requires and is incompatible with the memory envelope of edge hardware targeted by this work.

9. Conclusion

We have described an efficient two-branch pipeline for on-device plant disease perception, in which a lightweight decoder binary segmenter and a per-channel classifier operate independently on the same input frame. The segmenter uses addition based FPN skips at two scales and terminates in a single 8× nearest neighbor upsampling, eliminating the

transpose convolution decoder that normally dominates edge memory. The "blobbing" introduced by this aggressive decoder is framed as a controlled overcoverage that is matched to the actuation granularity of a variable rate sprayer, rather than as a defect to be corrected.

A central conceptual contribution of the paper is the *task sufficient accuracy* argument. Because the mask is consumed by a threshold on the per plant severity ratio, not by a human inspecting individual pixels, the binary spray decision is shown to be robust to segmentation errors up to $\sim$10% for all plants whose true severity lies outside a narrow borderline zone around the threshold. The directional bias of blobbing (systematic overcoverage) further concentrates the residual error on the less costly side of the decision: false positive spray events rather than missed diseased plants.

Evaluated end to end on a 10 species, 34 class PlantSeg dataset, Stage 1 achieves a mean foreground recall of 0.889 (range 0.86 to 0.92), and Stage 2 achieves a mean classification accuracy of 0.871 (macro F1: 0.859). A supplementary evaluation on PlantVillage data lifts classification accuracy to 0.956, confirming that the PlantSeg dataset results are data limited rather than architecture limited. With a parameter footprint of $\approx$380 KB after Int8 quantization and a peak activation memory of $\approx$192 KB, the full pipeline is compatible with resource constrained agricultural edge hardware.

Author Contributions: Conceptualization, K.K.K.; methodology, K.K.K.; software, K.K.K. and A.T.; validation, K.K.K., A.T. and E.K.; writing—original draft, K.K.K.; writing—review and editing, S.P.M. and E.K.; supervision, S.P.M. and E.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The PlantSeg dataset used in this study is publicly available. Code and trained model weights are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

1. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. In Proceedings of the Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
2. Kethineni, K.K.; Tang, A.; Mohanty, S.P.; Kougianos, E. Chroma-Sense: A Memory-Efficient Plant Leaf Disease Classification Model for Edge Devices. In Proceedings of the 2025 IEEE Conference on Artificial Intelligence (CAI), 2025, pp. 1–6. https://doi.org/10.1109/CAI64502.2025.00065.
3. Wei, T.; Chen, Z.; Yu, X.; Chapman, S.; Melloy, P.; Huang, Z. A Large-Scale In-the-Wild Dataset for Plant Disease Segmentation. *Scientific Data* **2026**, *13*, 205. https://doi.org/10.1038/s41597-025-06513-4.
4. Mitra, A.; Vangipuram, S.L.T.; Bapatla, A.K.; Bathalapalli, V.K.V.V.; Mohanty, S.P.; Kougianos, E.; Ray, C. Everything You Wanted to Know About Smart Agriculture. *arXiv preprint arXiv:2201.04754* **2022**.
5. Zhang, X.; Cao, Z.; Dong, W. Overview of Edge Computing in the Agricultural Internet of Things: Key Technologies, Applications, Challenges. *IEEE Access* **2020**, *8*, 141748–141761.
6. Varghese, B.; Wang, N.; Barbhuiya, S.; Kilpatrick, P.; Nikolopoulos, D.S. Challenges and Opportunities in Edge Computing. In Proceedings of the Proc. IEEE SmartCloud, 2016, pp. 20–26.
7. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861* **2017**.
8. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for MobileNetV3. *arXiv preprint arXiv:1905.02244* **2019**.
9. Tan, M.; Le, Q.V. EfficientNetV2: Smaller Models and Faster Training. *arXiv preprint arXiv:2104.00298* **2021**.

10. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv preprint arXiv:1707.01083* **2017**.
11. Li, Y.; Nie, J.; Chao, X. Do we really need deep CNN for plant diseases identification? *Computers and Electronics in Agriculture* **2020**, *178*, 105803.
12. Wei, K.; Chen, B.; Zhang, J.; Fan, S.; Wu, K.; Liu, G.; Chen, D. Explainable Deep Learning Study for Leaf Disease Classification. *Agronomy* **2022**, *12*.
13. Hughes, D.P.; Salathé, M. An Open Access Repository of Images on Plant Health to Enable the Development of Mobile Disease Diagnostics. *arXiv preprint arXiv:1511.08060* **2015**.
14. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proceedings of the Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015.
15. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In Proceedings of the Proc. European Conference on Computer Vision (ECCV), 2018.
16. Kethineni, K.K.; Mohanty, S.P.; Kougianos, E.; Bhowmick, S.; Rachakonda, L. SprayCraft: Graph-Based Route Optimization for Variable Rate Precision Spraying. *arXiv preprint arXiv:2412.12176* **2024**.
17. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv preprint arXiv:1801.04381* **2019**.
18. Tan, M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv preprint arXiv:1905.11946* **2019**.
19. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size. *arXiv preprint arXiv:1602.07360* **2016**.
20. Lin, J.; Chen, W.M.; Lin, Y.; Cohn, J.; Gan, C.; Han, S. MCUNet: Tiny Deep Learning on IoT Devices. In Proceedings of the Advances in Neural Information Processing Systems, 2020, Vol. 33, pp. 11452–11464.
21. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 248–255.
22. Mohanty, S.P.; Hughes, D.P.; Salathé, M. Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science* **2016**, *7*, 1419.
23. Archana, U.; Khan, A.; Sudarshanam, A.; Sathya, C.; Koshariya, A.K.; Krishnamoorthy, R. Plant Disease Detection Using ResNet. In Proceedings of the Proc. International Conference on Inventive Computation Technologies (ICICT), 2023, pp. 614–618.
24. Ferentinos, K.P. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture* **2018**, *145*, 311–318.
25. Ramcharan, A.; Baranowski, K.; McCloskey, P.; Ahmed, B.; Legg, J.; Hughes, D.P. Deep Learning for Image-Based Cassava Disease Detection. *Frontiers in Plant Science* **2017**, *8*, 1827.
26. Peker, M. Multi-channel capsule network ensemble for plant disease detection. *SN Applied Sciences* **2021**, *3*, 707.
27. Rakib, A.F.; Rahman, R.; Razi, A.A.; Hasan, A.S.M.T. A Lightweight Quantized CNN Model for Plant Disease Recognition. *Arabian Journal for Science and Engineering* **2024**, *49*, 4097–4108.
28. Bedi, P.; Gole, P. Plant Disease Detection Using Hybrid Model Based on Convolutional Autoencoder and Convolutional Neural Network. *Artificial Intelligence in Agriculture* **2021**, *5*, 90–101.
29. Chowdhury, M.E.H.; Rahman, T.; Khandakar, A.; Ayari, M.A.; Khan, A.U.; Khan, M.S.; Al-Emadi, N.; Reaz, M.B.I.; Islam, M.T.; Ali, S.H.M. Automatic and Reliable Leaf Disease Detection Using Deep Learning Techniques. *AgriEngineering* **2021**, *3*, 294–312.
30. Ashwinkumar, S.; Rajagopal, S.; Manimaran, V.; Jegajothi, B. Automated Plant Leaf Disease Detection and Classification Using Optimal MobileNet Based Convolutional Neural Networks. *Materials Today: Proceedings* **2022**, *51*, 480–487.
31. Barbedo, J.G.A. Plant disease identification from individual lesions and spots using deep learning. *Biosystems Engineering* **2019**, *180*, 96–107.

32. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proceedings of the Advances in Neural Information Processing Systems, 2015, Vol. 28, pp. 91–99.
33. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the Proc. IEEE/CVF CVPR, 2016, pp. 779–788.
34. Yu, C.; Wang, J.; Peng, C.; Gao, C.; Yu, G.; Sang, N. BiSeNet: Bilateral Segmentation Network for Real-Time Semantic Segmentation. In Proceedings of the Proc. European Conference on Computer Vision (ECCV), 2018.
35. Poudel, R.P.K.; Liwicki, S.; Cipolla, R. Fast-SCNN: Fast Semantic Segmentation Network. In Proceedings of the Proc. British Machine Vision Conference (BMVC), 2019.
36. Mehta, S.; Rastegari, M.; Caspi, A.; Shapiro, L.; Hajishirzi, H. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation. In Proceedings of the Proc. European Conference on Computer Vision (ECCV), 2018.
37. Alam, F. Leaf Disease Segmentation Dataset. Kaggle, 2023.