
# Assessment Group Report

Machine Vision (UFMFRR-15-M)

January 14th, 2025

# Contents

# 1 Introduction

Accurate apple counting is essential for orchard management, enabling precise yield estimation, resource allocation, and optimizing harvesting schedules. Effective fruit counting supports farmers in maximizing productivity and minimizing waste, making it a critical task in modern agricultural practices. Apples in orchards exhibit distinct features that facilitate their identification. Red apples typically display a predominantly red hue with characteristic yellow patches, whereas green apples maintain a nearly uniform green coloration. Unlike citrus fruits, apples possess a matte surface rather than a shiny appearance. Morphologically, apples do not conform to perfect spheres or regular polyhedrons; instead, they display a unique, irregular shape that varies subtly between individual fruits. Additionally, apples have a smooth texture, which contrasts with other orchard fruits that may have rough or dimpled surfaces. This study aims to develop a classical image processing pipeline specifically tailored to count red apples in orchard settings, excluding green varieties to streamline the analysis. The proposed method will be benchmarked against a state-of-the-art deep learning model to evaluate performance differences. All experiments are conducted using static images to focus on image-based techniques without the added complexity of video frame processing. By comparing traditional and modern approaches, this research seeks to determine the most effective strategy for accurate and efficient apple counting, ultimately contributing to enhanced orchard management practices.

# 2 Related Works

Most of the previous works exploited either few or all the characteristic features of apples mentioned previously. With respect to color, although the binarization of foreground and background may seem obvious in RGB space, particularly when dealing with red apples, in practice it is quite complex, because the leaves surrounding those apples also have a considerable red component althought they mostly appear green. However, in other color spaces, this distinctive feature could be more differentiable.

In [1], the idea involves acquiring RGB images and applying color thresholding in the La*b* color space to segment apple fruits from the background. The segmented binary images are further processed using the Circular Hough Transform (CHT) to detect circular shapes representing apples, including partially occluded ones. Finally, the number of apples is counted by analyzing the detected circular features. The CieLab color space (L*a*b*) separates an image into luminosity, green-red channel ($a^*$), and blue-yellow channel ($b^*$). As red apples generally are majorly red with tiny yellow patches, a higher $a^*$ and $b^*$ threshold values automatically segment apples from the leaves.

However, from the shape point of view, the Circular Hough Transform is not an optimal technique, as it iterates over all possible radii and edge points, performing a voting mechanism to identify circle centers. This can lead to significant processing time, especially for high-resolution images or large search ranges for radii. Since CHT relies heavily on edge detection, this method is susceptible to noise, and apples partially occluded by leaves don't form complete circles, making them hard to count.

Another color space worth noting is YCrCb, which splits RGB into luminance (L), red color difference ($C_r$), and blue color difference ($C_b$), as calculated in equation (1).

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

$$C_r = R - Y$$
$$C_g = G - Y$$
$$C_b = B - Y$$

In [2], 60 apple tree images captured under varying lighting conditions in an orchard were used to evaluate a methodology that enhances apple features through red color difference (Cr), computing optimal thresholds using maximum between-class variance, and segmenting apples effectively under these conditions. The method assumes consistently bimodal histograms of red color differences for accurate thresholding, but complex lighting conditions such as backlighting, shadows, or diffused light can result in unimodal or noisy histograms, leading to segmentation errors.

The methodology presented in [3] transforms RGB images into the YCbCr color space and applies texture and color-based segmentation to isolate mango fruits. Filters such as the Normalized Difference Index (NDI), variance thresholding, and Cr/Cb channel segmentation are iteratively applied, followed by particle analysis for fruit counting. Optimal thresholds for segmentation were determined through empirical testing and visual inspection, with iterative adjustments to refine differentiation between fruit and background. Instead of using a benchmark dataset, 593 manually annotated mango tree images, captured under varied real-world orchard lighting conditions, were utilized for evaluation.

The work presented in [4] utilizes the Fast Fourier Transform (FFT) leakage property to distinguish green citrus fruits from leaves by analyzing gray-scale variations via a circular grid, with binary preprocessing and morphological operations identifying probable fruit centers. Thresholds for FFT leakage, gray-scale intensity, and morphological parameters were empirically determined using 11 training images, with adjustments made for varying lighting conditions. The algorithm was validated on 60 images from a real-world citrus grove, achieving 82.2% detection accuracy. While novel in exploiting fruit surface symmetry and reflective properties, the approach struggles with detecting non-spherical objects like apples, which lack the consistent curvature required for FFT leakage analysis. Additionally, its reliance on fixed window sizes, sensitivity to lighting variability, and need for high-resolution images further constrain scalability and robustness, limiting its applicability to diverse orchard settings.

Similarly [5] converts RGB images to OHTA color space, utilizing $I'_2$ and $I'_1$ features for fruit extraction, and employs an improved Otsu adaptive threshold algorithm for segmentation under complex agricultural backgrounds. Just like in [2], the optimal threshold here was computed by maximizing between-class variance in the histogram of selected OHTA features, enabling robust one-dimensional segmentation. A dataset with 80 images of four fruit types (strawberry, tomato, pomegranate, and persimmon) captured under varying real-world agricultural conditions were used for validation.

Focusing on the texture and shape features of the apples, [6] presented an approach combining the eigenfruit approach, circular Gabor texture analysis, and color histogram-based background elimination to detect green citrus in natural canopies. Thresholds for eigenfruit classifiers and texture analysis were empirically derived using 32 training images, with adaptive thresholds modeled via regression. A dataset of 96 citrus grove images (32 for training, 64 for validation) was used for evaluation.

The methodology in [7] employs HSV color space conversion, color filtering, morphological operations, and contour detection. Optimal thresholds are derived experimentally for robust binarization. Polygon fitting uses least-squares approximation to simplify contours.

Green apples are identified by merging HSV masks for red and green. While effective, the approach struggles with occlusion and precise distinction from leaves, requiring refined filtering and shape analysis to improve accuracy. In [8], the approach involves detecting apples in orchard RGB images under natural light using a four-step algorithm: identifying potential apple pixels via color and smoothness, growing connected regions (seed areas), segmenting contours into arcs, and combining arcs into circles using a heuristic apple model. Dataset 1 used automatic exposure, while Dataset 2 utilized manually underexposed, sunset images. False detections were minimized by comparing arcs to ideal apples, achieving up to 95% accuracy under optimal lighting conditions.

From the perspective of Deep Learning, the introduction of CNNs helped solve multiple challenges faced by traditional methods. [9] deployed deep learning fruit detection using CNN to detect apples under different kinds of illuminations and occlusions which proved to be robust with a precision of 84.5%. This was the groundwork for the YOLO algorithm. YOLO (You Only Look Once) is an object detection model which predicts bounding boxes and probabilities of class in a single look at the image. [10] demonstrated the speed and accuracy of YOLO compared to region-based methods. This improved efficiency making it a perfect choice for real-time applications. Considering the application of apple detection and counting in particular, Yolov8 proved to be advancing over time with studies. Lightweight algorithms like Yolov8s-CFB [11] for real-time apple detection in complex environment provides high precision and efficiency in detecting. Integrating synthetic data with yolov8 made it more robust to detect apples in a natural surroundings with high accuracy. Multiple variants of Yolo including v8, v9, v10 and v11 were evaluated [12] for detection and counting fruits in complex environment for improving the performance. Yolo11n emerged as the fastest model with an inference speed of 2.4ms. Similarly, [13] utilized synthetic data augmentation to address the lack of labeled fruit images which achieved enhanced detection accuracy.

In this study, Yolov8 architecture was selected due to its speed and accuracy for real-time applications like apple counting and detection. Transfer learning was used in this methodology where the pre-trained YOLO model was fine-tuned with our custom dataset to detect apples precisely and accurately as mentioned. [14] introduced the evaluation metrics such as mAP, recall, and F1 score to evaluate the performance of these models which can be used to compare and improve different models and achieve higher accuracy.

# 3 Data Acquisition and Datasets

The primary dataset used for this research is the Minne Apple Dataset [15], a comprehensive collection designed for apple detection and segmentation tasks. This dataset consists of 1,000 high-resolution images (1280 x 720 pixels), collected over a period of two years, which contains 41,000 meticulously annotated labels. The annotations were created using the VGG annotator tool, involving the drawing of polygon masks to precisely capture the shape of fully or partially visible apples in each image. These objects are classified into three categories based on their pixel area: small ($< 322$ pixels), medium (between 322 and 962 pixels), and large ($> 962$ pixels). Importantly, apples located on the ground were excluded from the labeling process to focus solely on those visible on the tree.

Each image in the dataset contains up to 120 object instances, providing a diverse and challenging range of scenarios for apple detection algorithms. The manual annotation

process ensured a high level of accuracy, with every label verified for consistency and correctness. The dataset serves as a robust benchmark for developing and evaluating image processing and deep learning models, particularly in orchard environments.

In addition to the Minne Apple Dataset, this study incorporates a combination of other datasets to enhance the detection and classification of apples. Specifically, the task is focused on two distinct apple classes: Red apples and Green apples. The detection data from the Minne Apple Dataset was leveraged for this purpose. The process involved extracting the binary mask images and applying contour detection techniques to obtain the bounding boxes for the apples in the input images. These bounding boxes were then used to create annotated images, which were saved as labels for training the model.

To further improve the model's accuracy, additional datasets from Kaggle were integrated into the training process. These include:

- Apple and Orange Yolo Detection [16],
- Apple Detection Dataset [17],
- AppleBBCH81 Dataset [18].

These external datasets provide valuable test and validation data, complete with pre-annotated labels for all images. They introduce a wide range of variability in terms of illumination, apple appearances, and various scenarios, contributing to a more robust and accurate detection model. The final dataset, consisting of both red and green apples, forms a diverse and comprehensive foundation for training and evaluating apple detection models.

# 4 Methodology

## 4.1 Approach A - Classical Image Processing (CIP)

This outlines the methodology employed for the (CIP) approach to detect and count apples in orchards. The CIP pipeline improves classical image processing techniques, including color space transformations, thresholding, morphological operations, and contour-based analysis. Decisions regarding specific methodologies were made based on their effectiveness in related studies, as referenced below, and their suitability for the characteristics of our target dataset.

### 4.1.1 Image Standards and Input

High-resolution RGB images of apple orchards were captured under varying lighting conditions to simulate realistic scenarios. The dataset consisted of images with both occluded and non-occluded apples, diverse backgrounds, and varying apple sizes. This variability was crucial for evaluating the robustness of the proposed pipeline. The pipeline proceeds as shown in Figure 1, starting from loading an RGB image, since open cv reads images as BGR, the image is converted to RGB.

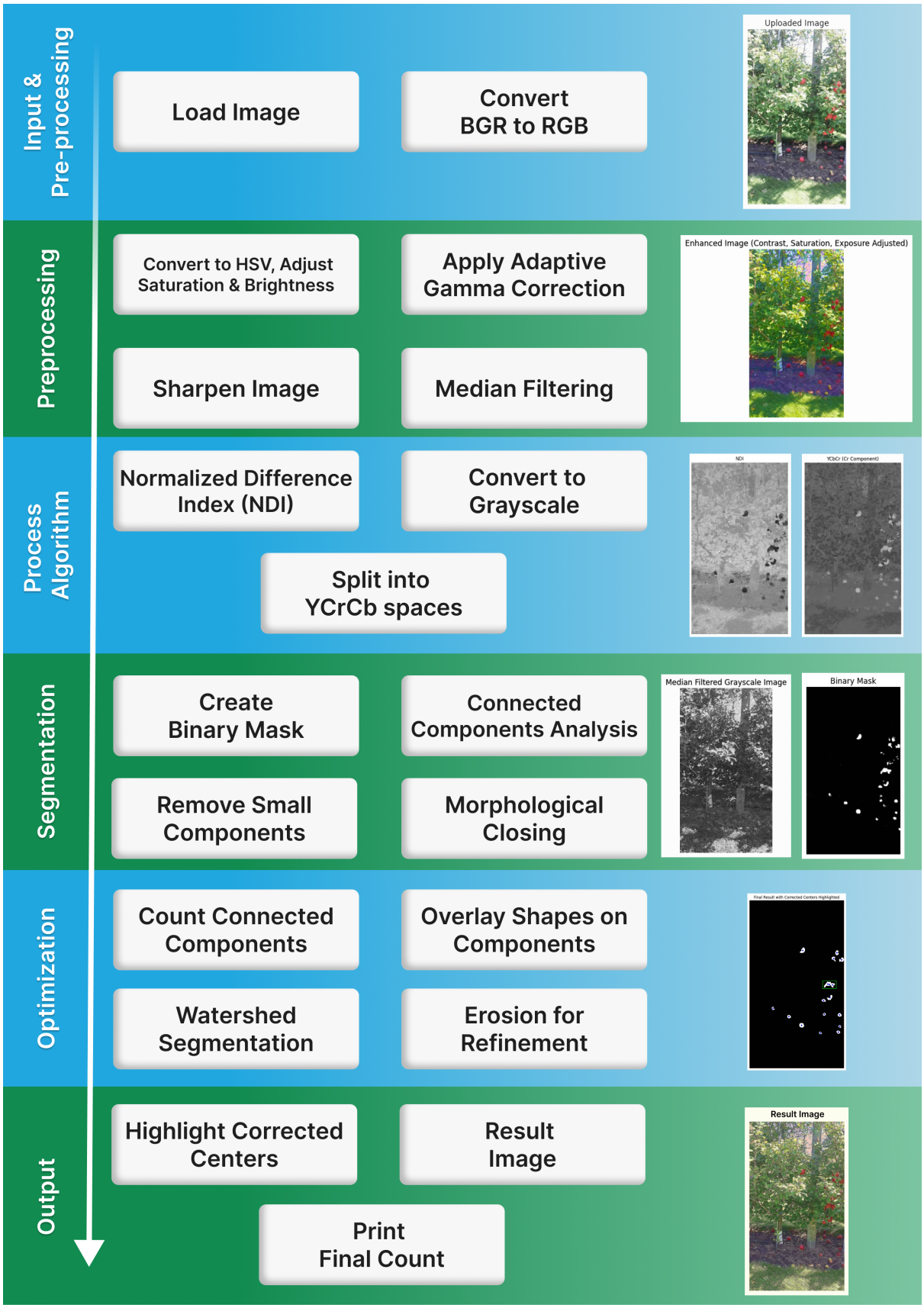



Figure 1: Methodology of CIP Approach

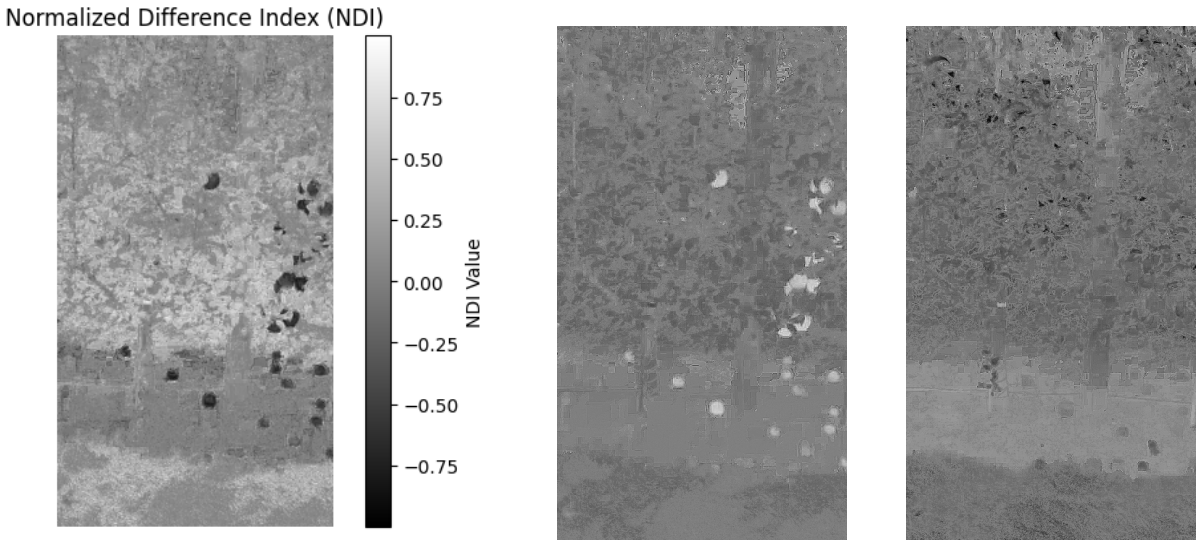
### 4.1.2 Pre-processing

The preprocessing step aimed to enhance the visual features of apples for subsequent processing stages. Based on insights from [1] and [3], the RGB image was first converted to HSV where the limits for saturation and value are limited within [50, 255] and normalized. This modified HSV image is then converted back to RGB and enhanced with Adaptive Gamma Correction thereby adjusting the brightness and contrast of the image based on its average intensity. The values selected were

$$(\gamma_{\text{low}} = 0.8, \quad \gamma_{\text{high}} = 1.5)$$

The enhanced image is sharpened to enhance object overlaps, edges, and corners by amplifying high-frequency details, improving feature distinction. It also accentuates textures, and few fine details, which can aid in feature extraction but may require denoising for cleaner results. Hence, this step is followed by a Median Blur implemented using a (7, 7) kernel.

### 4.1.3 Main Pipeline

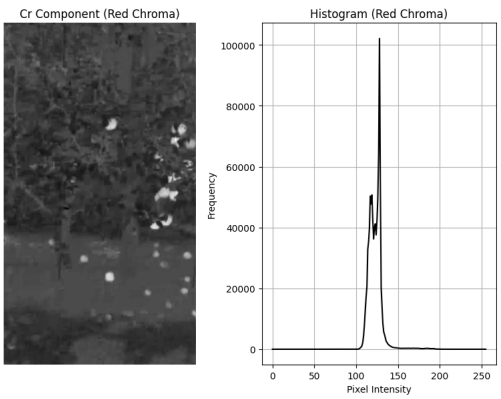



(a) NDI Image (b) Cr Component (c) Cb Component

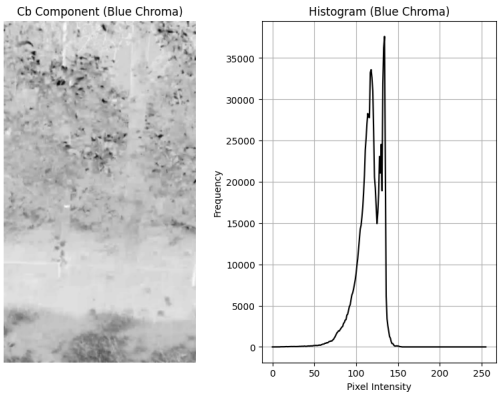
Figure 2: Sample Images for understanding

The main process begins by computing the Normalized difference Index (NDI) [3] as per equation (2), this step clearly highlights the red portions (apples) and green portions (leaves) in the image. Next, the sharpened image from the pre-processing step is converted to YCrCb color space using equation (1) and split into their respective channels. From Fig (2a) it can be observed that, for segmenting apples from its surroundings, the NDI value has to be negative, similarly from Fig (2b) and (3a) a threshold value of 160 for histogram of the Cr component seems sensible. A logical AND operation on these two criteria is used to binarize the image and segment apples from the background. It's worth noting that from Fig (2c) and Fig (3b) the threshold for Cb component doesn't add any value, hence it's ignored.

$$NDI = \frac{\text{Green} - \text{Red}}{\text{Green} + \text{Red}} \tag{2}$$




(a) Cr component




(b) Cb Component

Figure 3: Histograms of sample images

#### 4.1.4 Segmentation

After the generation of binary image, smaller patches (pixel area $< 250$) are removed through connected components so that the significant patches can be retained. Everywhere in the script, 8-point connectivity has been used. Onto these significant patches, in order to fill very small gaps within them a morphological closing operation is performed for just one iteration using a (5, 5) kernel, the result is called the Closed Binary Image. To segment any large patches (pixel area $> 1500$) as a result of overlapping fruits, a watershed algorithm is used as specified in Algorithm (1). Finally, Another round of connected components is run through the Updated Binary Image to find the final number of patches. The centroids of these patches are calculated and a bounding box is drawn for each patch based on its size. The centroids and bounding box coordinates are stored in two separate lists. The number of bounding boxes is counted as it symbolically represents the number of fruits in an image, as can be observed in Fig (5b). The effectiveness of the count was validated against ground truth annotations.

#### 4.1.5 Why These Decisions Were Made

The choices for specific methods were driven by:

1. **Robustness Against Variability:** Pre-processing techniques like image contrasting and Adaptive Gamma correction were selected for their proven ability to handle varying lighting.

2. **Focus on Red Apples:** The Cr channel and NDI thresholding were prioritized as they directly red and green target hues [1], [2], [3].

3. **Overlap Handling:** Watershed segmentation was included to address the challenge of clustered apples as indicated by large patches in the binary image.

4. **Other Considerations:** The size of apples in the Minne Apple dataset [15] are quite small, at such level it is difficult yo analyze texture properties of the fruit, hence the complete pipeline involved segmentation based on color and patch size rather than textures like done in [6].

**Algorithm 1** Optimized Segmentation

```
if patch_area ≥ 1500 then
    centroid, h, w ← getPatchCoords()
    drawGreenRect(centroid, h, w)
    ROI ← closed_binary_image(centroid, h, w)

    if size(ROI) is 0 OR None then
        return closed_binary_mask
    end if

    dist_transform ← computeDistanceTransform(ROI, L2_distance)

    if dist_transform is 0 OR None then
        return closed_binary_mask
    end if

    foreground ← threshold(dist_transform)
    background ← dilate(ROI, kernel)
    unknown ← subtract(background, foreground)
    markers ← connectedComponents(foreground)
    watershed(ROI, markers)

    roi_split ← erode(ROI, kernel)
    closed_binary_image(centroid, h, w) ← roi_split

    return closed_binary_image
end if
```

The CIP approach integrates customized image processing techniques for reliable apple detection and counting, addressing complex backgrounds, occlusion, and lighting variability with a modular, adaptable pipeline.

## 4.2 Approach B - ML (Machine Learning)

Based on the literature review, it can be concluded that the Yolo familty is an excellent choice for object detection. Based on that YOLOv8 is considered here for fruit detection due to its efficient architecture and high accuracy in real-time object detection. The ML pipeline starts with data-preprocessing, followed by going through the architecture and a post-processing step to finally optimize the output.

### 4.2.1 Data Pre-processing

The YOLOv8 model requires a specific pre-processing pipeline to ensure that input images are in the right format and size. The following steps are carried out as part of the pre-processing:

- **Resizing:** Input images are resized to a fixed dimension of 640 x 640 pixels to match the input size expected by the YOLOv8 model.

- **Augmentation:** Random flipping, color jitter, and rotation are applied to augment the dataset. This helps in enhancing the model's robustness by exposing it to various image orientations, and scales allowing the model to generalize better in real-world scenarios.
- **Normalization:** Images are normalized to a specific range to reduce the influence of pixel intensity variations.

#### 4.2.2 Main Pipeline

YOLOv8 employs an anchor-free approach, eliminating the inefficiencies associated with anchor-based detection methods. This improves the learning speed. The complete architecture of Yolov8 can be found in Fig (4). It consists of a backbone, neck and a head which are detailed below,

#### 4.2.3 Backbone

The backbone serves as the feature extractor in the YOLOv8 architecture, capturing both high-level and low-level features to provide a rich hierarchical representation of the input image. The architecture employs the SiLU activation function, which is mathematically defined as:

$$\text{SiLU}(x) = x \cdot \sigma(x), \tag{1}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

- **Output Range:** $[-\infty, +\infty]$.
- **Non-Linearity:** The sigmoid part of the function adds non-linearity to the system, enabling the modeling of complex patterns from data provided.
- **Handling Negative Values:** Unlike ReLU, this can handle negative values, preserving information from negative inputs from previous layers.
- **Gradient Flow:** Its smooth gradient improves gradient flow during backpropagation, making it particularly effective in capturing subtle details in images and leading to better convergence and accuracy.

#### 4.2.4 Neck

The neck is the bridge connecting the backbone and the head, it collects and assembles feature pyramids by aggregating feature maps. It performs feature fusion operations while integrating contextual information. Additionally, it reduces spatial dimensionality to improve computation speed with minimal compromise in performance.

#### 4.2.5 Head

The head generates outputs in the form of bounding boxes with confidence scores and class labels. These tasks are done separately (i.e decoupled), which could lead to misalignment problems. To address this, YOLOv8 introduces a task alignment score, computed as the product of the classification score and the Intersection over Union (IoU) score. This score is used to identify positive and negative samples.

- **Positive Sample Selection:** The model selects top-k positive samples based on the alignment score.
- **Loss Functions:**
  - **Binary Cross-Entropy (BCE):** Measures the difference between true and predicted labels.
  - **Complete IoU (CIoU):** Accounts for bounding box center and aspect ratio alignment with ground truth.
  - **Distributional Focal Loss (DFL):** Refines boundary predictions by emphasizing false-negative corrections.

The architecture of YoloV8 is optimized to predict objects of various sizes (small, medium and large), hence making it perfect for settings like the Minne Apple dataset.

#### 4.2.6 Hardware and Software Setup

- **Hardware:** NVIDIA RTX3090 Ti (24 GB), 12-core CPU, and 64 GB RAM
- **Operating System:** Ubuntu 22.04
- **Dependencies:** Ultralytics YOLO library, CUDA, and Python libraries

#### 4.2.7 Post-Processing

Non-Maximum suppression (NMS) technique enhances the efficiency and accuracy of detecting an object. During prediction, multiple bounding boxes for the same apple could be created due to the nature of Yolo being a region-based search. As a post-processing step, extra NMS is implemented to remove twin or identical bounding boxes for the same apple. Post-processing is applied to filter out false positives based on a confidence threshold of 0.5.

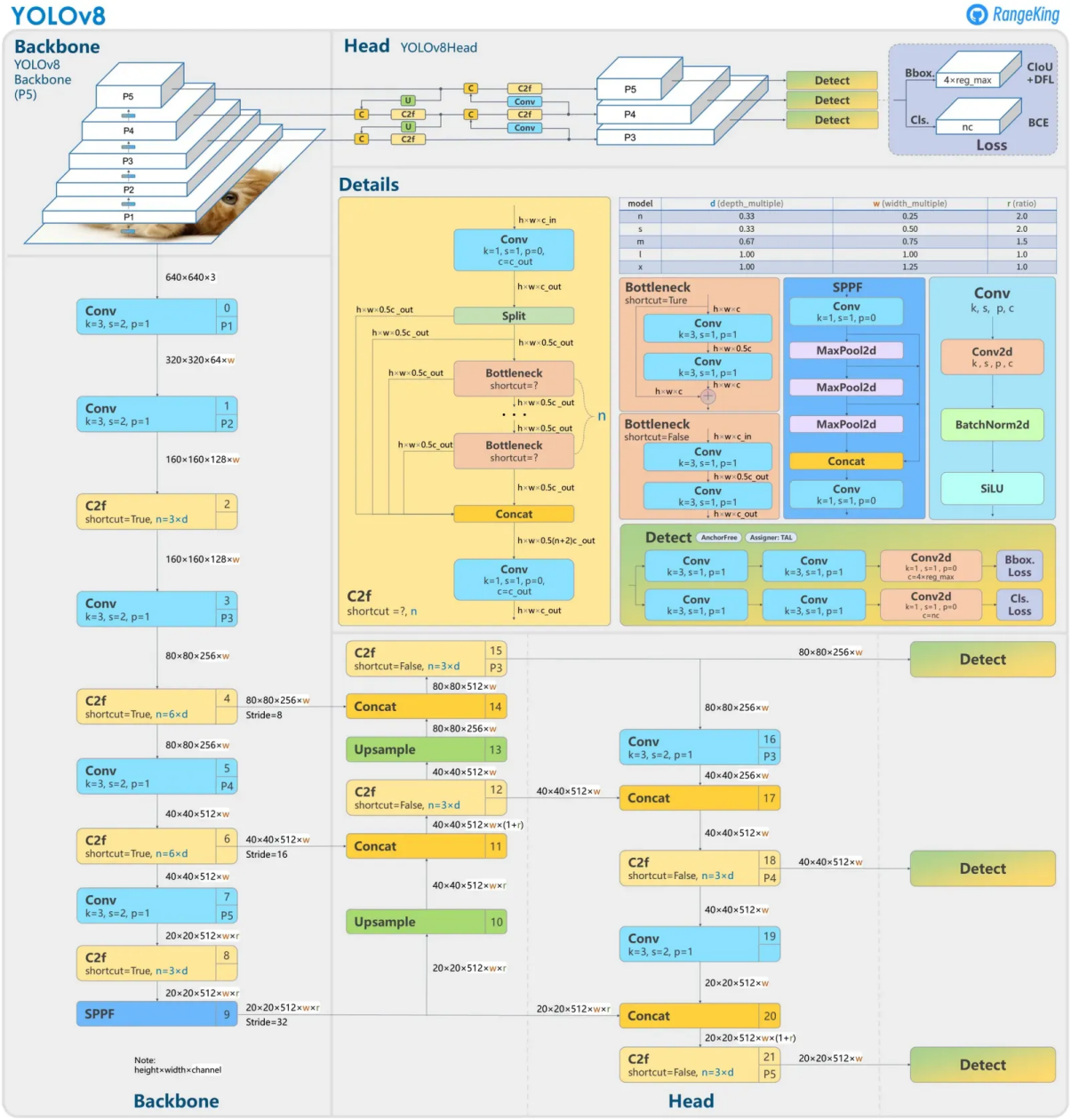



Figure 4: Methodology behind YOLO architecture (Image Source: [19])

# 5 Experiment and Implementation

## 5.1 CIP (Classical Image Processing) Approach

The Classical Image Processing (CIP) method focuses on detecting apples using traditional computer vision techniques. This approach leverages the power of image processing algorithms and mathematical operations to identify apples in a given image. In this section, we outline the key steps involved in the CIP-based approach for apple detection.

The **detections** directory of the minne apple test set has been used for testing. It consists of 331 images with ground given in a JSON file. These ground truth labels were converted to YOLO format for every single image. The testing script, runs the

whole image processing pipeline (from loading the image to getting the bounding box coordinates) for every single image and compares its predictions with the ground truth to calculate true positives, false positives, and false negatives as follows,

- A **True Positive (T+)** occurs if the predicted bounding box and the ground truth box overlap with IoU $\geq 0.25$.
- A **False Positive (F+)** occurs if the predicted bounding box does not overlap with any ground truth box, i.e., IoU $< 0.1$.
- A **False Negative (F-)** occurs if a ground truth box exists but no corresponding predicted box is found.

These metrics are calculated for each image in the test set, and the results are stored in a CSV file along with the image name. Precision (Pi) and Recall (Ri) are then calculated for each image using the following equations:

$$\text{Precision } (\text{P}_\text{i}) = \frac{T^+}{T^+ + F^+}, \quad \text{Recall } (\text{R}_\text{i}) = \frac{T^+}{T^+ + F^-}$$

Finally, the **mean Average Precision (mAP)** and **mean Average Recall (mAR)** are computed across all images in the dataset as below, to evaluate the overall performance of the CIP approach.

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} P_i, \quad \text{mAR} = \frac{1}{N}\sum_{i=1}^{N} R_i$$

## 5.2 ML (Machine Learning) Approach

In contrast to the CIP approach, the Machine Learning (ML) approach relies on deep learning models to learn patterns from data and make predictions about apple detection. In this work, we employed the YOLOv8 (You Only Look Once version 8) model, a state-of-the-art deep learning model known for its efficiency and speed in object detection tasks.

### 5.2.1 Model Training

The model is trained using pre-trained weights ('Yolov8s.pt') that have been fine-tuned on large-scale object detection datasets. The YOLOv8 architecture has various variants, with 'Yolov8s' being chosen for its computational efficiency and reliability for real-time applications such as apple detection. The training process follows these steps:

- **Data:** The training data includes the combination of the 4 datasets mentioned previously in which 90% of the images were used for training and the images were of two classes namely Red Apples and Green Apples.
- **Batch Size and Epochs:** The model was trained with 200 epochs and the training duration was approximately 48 minutes. A batch size of 32 was chosen to balance memory usage and training time. Its important to note that shuffle was disabled during the whole training process.

- **Input Configuration:** A YAML configuration file specifying the paths to the training and validation data, the number of classes, and the image size (640 x 640) are used to configure the training process.
- **Fine-Tuning:** The model is fine-tuned by using the best-performing epoch from the previous session, ensuring that the learning process builds on the most optimal parameters.

Throughout the training phase, metrics and loss values were logged after each epoch. The learning rate was automatically adjusted using a cosine annealing scheduler to maintain a high learning rate initially and gradually decrease it for fine-tuning the model.The overall training process is summarized in Algorithm (2).

**Algorithm 2** YOLOv8 Training, Evaluation, and Testing

```
Input: "yolov8s.pt" and "apple.yaml"
Output: Trained model and Validation metrics

Initialize model: model ← yolov8s.pt"
Load dataset: data ← apple.yaml

Train the model:
    model.train(data, epochs=200, batch=32, imgsz=640)

Evaluate model performance:
    val_results ← model.val()
    print(val_results)

Test the model:
    test_image ← test_images
    results ← model(test_image, save=True, save_txt=True)
```

### 5.2.2 Validation and Testing

Once trained, the model is validated using the rest 10% of the training set. The validation process helps identify any overfitting or underfitting issues. Multiple metrics were measured to evaluate model performance during validation, including:

- *mAP@0.5*: The mean Average Precision (mAP) with an Intersection over Union (IoU) threshold of 0.5.
- *mAP@0.5:0.95*: The mAP averaged over IoU thresholds ranging from 0.5 to 0.95, increased with step size of 0.05.
- *Precision* and *Recall*: These metrics are crucial for evaluating the ratio between correct detections and false positives.

A high mAP value above 90% indicates excellent detection accuracy, while values below that suggest room for improvement. Trends in model performance were visualized using graphs. Subsequently, test images are run through the model, and apples are detected

with bounding boxes. Similar to the CIP method, the testing was done on the 331 images in the **detection** directory of the Test set in Minne Apple dataset. The same ground truth labels were converted to YOLO format for evaluation.

## 5.3 Combined Information for Both CIP and ML Approaches

Both the CIP and ML approaches follow a structured pipeline to detect apples, but they differ significantly in the methods used. While the CIP approach relies on hand-crafted image processing techniques, the ML approach leverages the power of deep learning models such as YOLOv8 for end-to-end learning.

### 5.3.1 Key Similarities:

- Both approaches rely on bounding boxes for apple detection and evaluate performance using Precision, Recall, mAP and mAR metrics.
- In both cases, performance is evaluated using ground truth annotations, and results are stored in a CSV file for further analysis.

### 5.3.2 Key Differences:

- **Pre-processing:** The CIP approach involves traditional image processing steps, such as noise removal and thresholding, while the ML approach focuses on augmentations like flipping, color jitter, and resizing.
- **Modeling:** The CIP approach does not use any learned model and instead applies classical vision techniques, whereas the ML approach utilizes a pre-trained deep learning model that adapts to the data during training.
- **Detection Method:** In CIP, apple detection is based on geometric features, edge detection, and heuristics, whereas ML employs YOLOv8 to automatically learn features from large datasets.

### 5.3.3 Evaluation Metrics Comparison:

Both methods calculate Precision, Recall, and mAP, but the ML approach typically provides better results due to its ability to learn complex patterns from data. However, the CIP approach offers a more interpretable solution, as it uses traditional computer vision techniques.

Both CIP and ML approaches have their strengths and weaknesses. While the CIP approach provides an efficient, interpretable solution for apple detection in simpler scenarios, the ML approach offers superior performance, particularly when dealing with complex and varied datasets. Future work may explore hybrid solutions combining the best aspects of both approaches to improve accuracy and robustness.

# 6 Results and Evaluation

(a) Input Image (b) Output Mask with bounding boxes

Figure 5: Input and Output over Sample image

In case of CIP, it was observed that even the fruits on the ground which had a considerable red tint were predicted, which is correct as the system works on color-based segmentation. However, in the Minne apple dataset, the apples on the ground and those hidden between leaves weren't labeled (the hidden apples were automatically ignored by the CIP pipeline). Due to this, there were too many False Positives, which eventually led to the result presented in Table 1.

| **Average $T^+$** | **Average $F^+$** | **Average $F^-$** | **mAP** | **mAR** |
|---|---|---|---|---|
| 11.85 | 11.28 | 25.27 | 0.55 | 0.36 |

Table 1: Results before adding the extra step for filtering ground-level predictions.

To counter this, an extra step was added while calculating the metrics. The predictions whose bounding box centroid was below the lowest fruit labeled in the ground truth were not considered valid predictions. Essentially, anything below the lowest fruit labeled in the ground truth was considered to be on the ground and ignored. After implementing this step, the number of False Positives in every image reduced drastically (from approximately 11 per image to around 3), and this also improved the mean average precision (mAP), as shown in Table 2.

| Average $T^+$ | Average $F^+$ | Average $F^-$ | mAP | mAR |
|---|---|---|---|---|
| 11.85 | 3.11 | 25.27 | 0.82 | 0.36 |

Table 2: Results after implementing the extra filtering step.

Figure 6: Zero $T^+$ prediction images Minne Apple.

Inspite of this correction, there were only four images in the test set that had zero true positive predictions. These images are listed as follows and shown in Figure (6) :

- Dataset4_front_120.png (top-left)
- Dataset2_back_900.png (top-right)
- Dataset2_back_960.png (bottom-left)
- Dataset2_back_1350.png (bottom-right)

.

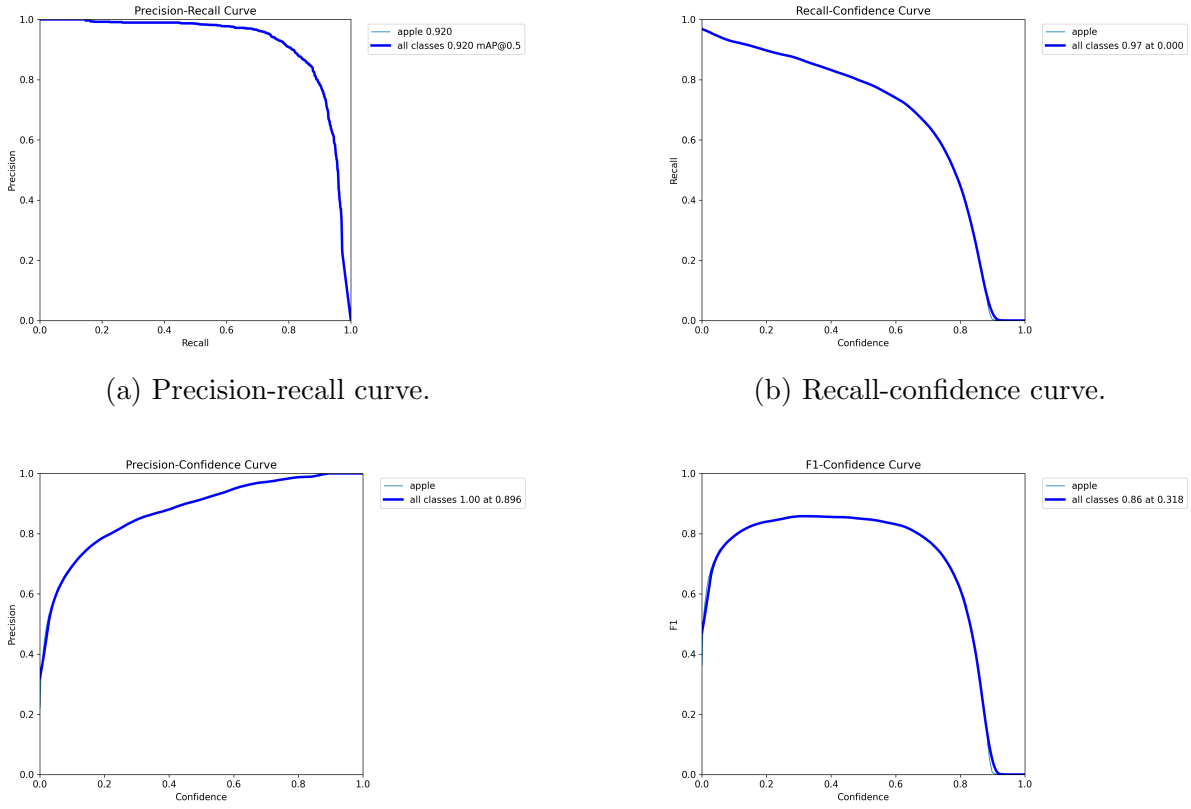
When verified, it was noted that each of these images either had very few ground truth labels or the apples in those images had very little red color, with larger yellow patches. Hence, it can be concluded that the classical image processing pipeline presented here is precise for counting red apples from orchards.

**Machine Learning approach**:
After the model has been successfully trained, validated and tested, we can now look into the validation results for more insight on the performance. From the set of figures below, PR curve (Fig. 7a) explains the relation between precision and recall for the model. It can be seen that precision decreases as recall increases, indicating that the model will find more true positives while sacrificing weight over false positives.

The recall-confidence curve (Fig. 7b) explains the relation between recall and confidence scores for the model. As the confidence increases, recall decreases, showing the model's ability to detect true positives and how it behaves at different confidence levels during object detection.

The precision-confidence curve (Fig. 7c) tells us the relation between precision and confidence. The ratio is proportionate, indicating that precision increases as the confidence of prediction increases.

Finally, Fig. 7d shows the F1 curve for the model. The F1 score reaches a peak where precision and recall are optimally balanced.




(a) Precision-recall curve.

(b) Recall-confidence curve.

(c) Precision-confidence curve.

(d) F1-confidence curve.

Figure 7: Comparison of precision-recall, recall-confidence, precision-confidence, and F1-confidence curves for the model.

During overfitting and underfitting, convergence will not be seen. From the graphs shown in Fig. 8, we can clearly see that the training and validation losses flatten out without any oscillations or errors, which means that the model has learned as much as possible. The graphs indicate that the model is converging at the end of training.

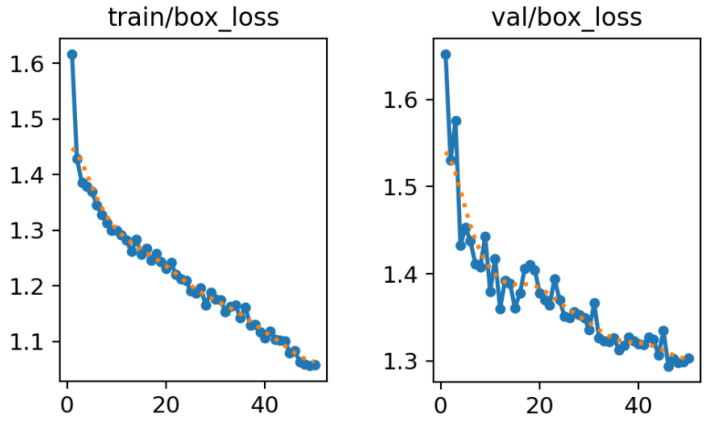



Figure 8: Training and validation box loss curves, showing the model's trend.

After the model has been successfully trained and validated, we can now look into the test results in Table 3 for more insight on the performance. These were a result of adding NMS and applying the ground constraint (predictions below the lowest ground truth label will be ignored). The mAP value is measured at 0.5 IoU criteria:

| **Average $T^+$** | **Average $F^+$** | **Average $F^-$** | **mAP** | **mAR** | **Average Inference Time** |
|---|---|---|---|---|---|
| 21.85 | 8.98 | 17.71 | 0.71 | 0.54 | 2.5 ms |

Table 3: Results before adding the extra step for filtering ground-level predictions.

Clearly YoloV8s has more true predictions compared to the CIP method. This is quite expected considering he fact that images in the Minne Apple dataset have really small apples which are neglected by the connected components analysis in the CIP pipeline, however Yolo on the other-hand is constructed to perform well and pick up even these small objects. Yolo outperforms CIP even with lesser False negatives leading to a higher Recall value which is quite obvious as well. However, Yolo has too many false positives compared to CIP, this is because Yolo accurate;y predicts the existance of most of the apples in the image, but for every single image in the Minne Apple dataset, only the apples on the tree right in front of the camera were labeled, basically most fruits in the background trees were ignored [15]. This can be observed in the few test images attached in the Fig (9).


Figure 9: This figure shows six output images with both predicted and ground truth annotations overlayed. The red boxes represent the predictions from the model, while the green boxes represent the ground truth annotations. The comparison highlights the model's performance in detecting apples from foreground and background.

# 7 Conclusions and Future Works

To conclude, this report considers two Machine Vision pipelines namely CIP and YoloV8. These two pipelines are evaluated against the benchmark Minne Apple Dataset (detections cagtagory). Both of them show convincing results based on the ideology behind their development. There's still room for improvement, in case of CIP, the recall value is

compromised because of a significant False negatives compelling improvements needed to detect very small objects accurately. YoloV8 on the other hand needs further pre or post processing steps to ignore fruits in the background. Both of these can be a significant improvement to this work in the future. An extension to this work might be is to apply these techniques in a real-world setting to evaluate the reliability of their performance.

# References


[1] S. K. Behera, N. Mishra, P. K. Sethy, and A. K. Rath, "On-tree detection and counting of apple using color thresholding and cht," in *2018 International Conference on Communication and Signal Processing (ICCSP)*, 2018, pp. 0224–0228.

[2] D. M. Bulanon, T. Kataoka, Y. Ota, and T. Hiroma, "Ae—automation and emerging technologies: a segmentation algorithm for the automatic recognition of fuji apples at harvest," *Biosystems engineering*, vol. 83, no. 4, pp. 405–412, 2002.

[3] A. B. Payne, K. B. Walsh, P. Subedi, and D. Jarvis, "Estimation of mango crop yield using image analysis–segmentation method," *Computers and electronics in agriculture*, vol. 91, pp. 57–64, 2013.

[4] R. Bansal, W. S. Lee, and S. Satish, "Green citrus detection using fast fourier transform (fft) leakage," *Precision Agriculture*, vol. 14, pp. 59–70, 2013.

[5] X. Wei, K. Jia, J. Lan, Y. Li, Y. Zeng, and C. Wang, "Automatic method of fruit object extraction under complex agricultural background for vision system of fruit picking robot," *Optik*, vol. 125, no. 19, pp. 5684–5689, 2014.

[6] F. Kurtulmus, W. S. Lee, and A. Vardar, "Green citrus detection using 'eigenfruit', color and circular gabor texture features under natural outdoor conditions," *Computers and Electronics in Agriculture*, vol. 78, no. 2, pp. 140–149, 2011.

[7] J. Dong, F. Xue, T. Dong, and H. Tang, "Research on apple recognition and counting model based on image processing," in *2024 5th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, 2024, pp. 120–124.

[8] R. Linker, O. Cohen, and A. Naor, "Determination of the number of green apples in rgb images recorded in orchards," *Computers and Electronics in Agriculture*, vol. 81, pp. 45–57, 2012.

[9] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, "Deepfruits: A fruit detection system using deep neural networks," *Sensors*, vol. 16, no. 8, 2016. [Online]. Available: https://www.mdpi.com/1424-8220/16/8/1222

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[11] R. M. Y. Z. . J. G. Bing Zhao, Aoran Guo, "Yolov8s-cfb: a lightweight method for real-time detection of apple fruits in complex environments." *Real-Time Image*, vol. 21, 08 2024.

[12] R. Sapkota, Z. Meng, M. Churuvija, X. Du, Z. Ma, and M. Karkee, "Comprehensive performance evaluation of yolo11, yolov10, yolov9 and yolov8 on detecting and counting fruitlet in complex orchard environments," 2024. [Online]. Available: https://arxiv.org/abs/2407.12040

[13] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[14] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[15] N. Hani, P. Roy, and V. Isler, "Minneapple: A benchmark dataset for apple detection and segmentation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, p. 852–858, Apr. 2020. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.2965061

[16] H. Fakher, "apple and orange yolo detection dataset," https://www.kaggle.com/datasets/hossamfakher/apple-and-orange-yolodetection/data, 2022, accessed: 2025-12-10.

[17] S. Mahata, "Apple detection dataset," https://www.kaggle.com/datasets/snehamahata/apple-quality-dataset, 2024, accessed: 2025-12-11.

[18] S. Kodors, I. Zarembo, G. Lacis, L. Litavniece, I. Apeinans, M. Sondors, and A. Pacejs, "Autonomous yield estimation system for small commercial orchards using uav and ai," *Drones*, vol. 8, p. 734, 12 2024.

[19] Ultralytics, "Image from github issue #189," 2025, accessed: 2025-01-07. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189