UFMFRR-15-M Machine Vision

Group Report on Apple Counting in Orchards

Group: WED 1500 7

Oscar Siu 23080476
Samrat Ghosal 23080447
Sunil Thapa 23080489
Adharv Punathum Kandivil 23080495
Madhurya Mozumder 23080485
Manikandan Cheruvary 23080472

Group Members Contribution

No.	Name of group members	Contribution to project	Contribution to report	Signature
1	Oscar Siu	Machine Learning Method	Implementation, Results for ML, Conclusion, Consolidation	Oscar Siu
2	Samrat Ghosal	Traditional Method	Introduction, Data Acquisition & Dataset, Methodology, Experiment and Implementation, Result & Evaluation	Samrat Ghosal
3	Sunil Thapa	Traditional Method	Initial Programmes, Methodology, Literature Review and parts of Experiment and Implementation and Future Work.	Sunil Thapa
4	Adharv Punathum Kandivil	Traditional & Machine Learning Method	Introduction, Related Work, Data Acquisition and Dataset, Results for Traditional	Adharv Jagan
5	Madhurya Mozumder	Traditional Method	Data acquisition, experiment and implementation, Algorithms and parts of refining code.	Madhurya Mozumder
6	Manikandan Cheruvary	Machine Learning Method	Methodology and Implementation	Manikandan Cheruvary

Contents

Group Members Contribution.......... 2
Introduction.......... 4
Related Works.......... 6
    Traditional Method.......... 7
Methodology.......... 10
    A. Conventional Method.......... 10
    B. Machine Learning Method.......... 14
Experiment and Implementation.......... 18
    Machine Learning Method.......... 21
Results and Evaluation.......... 23
    Traditional Method.......... 23
    Comparing Conventional and Machine Learning Method.......... 28
Conclusion and Future Works.......... 32
References.......... 33

Introduction

This report compares the effectiveness of two distinct methodologies for apple counting in orchards: traditional Machine Vision approach and modern Machine Learning approach.

The traditional machine vision approach uses HSV colour separation techniques to analyse the image of the apple trees, identifying and counting individual apples on the trees based on characteristics namely colour and shape. The process involves using a dataset of 668 images of apple trees where the apples can be distinguished by their various characteristics like colour, shape, and texture. These visual cues are leveraged to segment the apples from the background and the apples present in the frame of interest are counted.

HSV stands for Hue, Saturation, and Value. Unlike the RGB colour model, which represents colours as a combination of red, green, and blue channels, the HSV colour model separates colour information into three distinct components. The hue component represents the dominant wavelength of light, determining the colour's perceived shade. Saturation refers to colour intensity. The higher saturation values, the more vibrant colours. Lastly, the value component represents the brightness or lightness of the colour.

There are various steps involved in the conventional method of apples counting, wherein the colour difference is leveraged to mask out the apples from the background. Contour detection is done, and upon calculation of the area occupied by each isolated region by the contour, the area is assigned to be of an apple. The image is further passed through a noise reduction filter. Finally, the counting algorithm is applied to quantify the number of apples present in the given image.

On the other hand, machine learning, especially when performing tasks such as apple counting in an environment, is a multi-step procedure that includes data acquisition, model selection, training and evaluation. For apple counting, we have a dataset of 668 images of green and red apples on trees in the orchard. These images are then labelled, by marking apples with bounding boxes. This step provides ground truth data that the model will learn from.

Secondly, the selection of the right machine-learning framework is essential. YOLO (You Only Look Once) is a popular choice as it is specifically known for its speed and efficiency. Other tools and libraries like Keras, PyTorch, and OpenCV are also frequently used, each offering unique features for various aspects of machine learning and image processing.

Lastly, the learning process is at the heart of machine learning, and models like Convolutional Neural Networks (CNN) are frequently employed for tasks like object detection. The annotated dataset is used to train the model, which learns to detect and count apples from the images. This training includes several iterations and requires a significant amount of processing power, where GPUs are essential.

The main objective of this project is to give a detailed comparison between both traditional and machine learning approaches and compare and investigate their effectiveness on counting apples in orchards.

Related Works

Machine Learning Method

"A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5" by Bin Yan, Pan Fan, Xiaoyan Lei, Zhijie Liu and Fuzeng Yang. [1]

The study proposes an apple target detection method for picking robots using an improved YOLOv5s model. The study involves collecting 1,214 images from Fuji apple trees under various environmental conditions for a comprehensive dataset. The methodology includes preprocessing 1,014 images for training by labelling, applying data augmentation techniques for robustness, and modifying the YOLOv5s architecture to optimize model size and efficiency. The model showed high accuracy. But it has limitations, such as not working at night and the inability to detect green apples.

A Two-Stage Deep Learning Model for Detection and Occlusion Based Classification of Kashmiri Orchard Apples for Robotic Harvesting by: Divya Rathore1, L. G. Divyanth1… [2]

The study introduces a two-stage machine-learning model for apple detection in orchards, utilizing YOLOv7 for detection and EfficientNet-B0 for occlusion-based classification. The first stage detects apples under various conditions using YOLOv7, tailored to a custom dataset of Kashmiri orchard images. The second stage classifies detected apples into occlusion categories using EfficientNet-B0, incorporating techniques like contrast stretching for accuracy. This approach effectively addresses both detection and classification challenges critical for robotic apple harvesting. However, its real-world efficacy might hinge on dataset diversity and model adaptability in different orchard environments. The approach may be further improved by capturing a larger dataset with green apples and nighttime conditions, aiming to enhance the model's applicability across different scenarios.

Apple Detection in Complex Scenes Using the Improved YOLOv4 Model by Lin Wu, Jie Ma, Yuehua Zhao and Hong Liu [3]

This paper presents an enhanced version of the YOLOv4 model, named EfficientNet-B0-YOLOv4 for apple detection in orchards, addressing the challenges posed by complex backgrounds and leaf occlusions. Apple images were collected using crawler technology and then labelled. To address the issue of insufficient data due to leaf occlusions, traditional data augmentation techniques were used alongside a novel leaf illustration data augmentation method. By using these augmented images for training, the model learns to recognize apples even when they are partially obscured by leaves. The model achieved high scores in key performance metrics: a Recall of 97.43%, mAP of 98.15%, and an F1 score of 96.54%.

Traditional Method

Object Tracking Using HSV Values and OpenCV Shafi, G., Garg, P. (2021) [4]

This paper presented the use of employing certain attributes of an object to be detected. The attribute chosen was RGB colour and it was then translated to HSV values using OpenCV as the utility of choice. In this paper, background subtraction was done, and noise elimination was done using median filtering. In the morphological processing part, dilatation and erosion were used to achieve morphological closure.

This paper essentially is for object tracking but the principles used here were employed by us for apple detection and it performed well. The paper is pertinent to our project as it employs the use of the HSV colour model we have used in our project. Furthermore, it has also used OpenCV just like the way we have done as our choice.

Automatic recognition vision system guided for apple harvesting robot Ji W., Zhao D., Cheng F., et al., (2011) [5]

This is one of the early papers that employed the use of the HSV colour space in the detection of Fuji variety of red apples in an orchard. The paper used the invariance of the round shape of the apples to its advantage thereby resulting in effective outlining of the apples' perimeter. Development of apple detection algorithm followed the seeded region growing and colour feature method. The classification algorithm uses the Support Vector Machine which used both HSV colour and round shape of the apples to identify them. This method, although high on effective accuracy is affected by tree foliage and is applicable only for green apples. Also, the images used in detection of apples usually have a clear close-up of apples which is different from the dataset we are using. The essential mechanism used in this paper is similar to the working logic we have used.

APPLE RECOGNITION BASED ON MACHINE VISION Jiang G., Zhao C. (2012) [6]

This paper presented the detection of apples in an orchard using colour image transformation by transforming colour image into grayscale image. Also, a technique called Hough transformation is used to find the centre and the radii of the apples. The recognition rate is high. However, the major drawback of this paper is that if Apple is not perfectly round, then detection might become very challenging due to the poor roundness. Also, this approach again does not work very well for apples whose photographs are taken from a distance.

To conclude, this paper depends upon the conversion of RGB colour to a different scale mechanism – in this case grayscale - for robust apple detection.

Data Acquisition and Datasets

Various kinds of hardware and technologies can be used to capture images of apples in an orchard.

- **Standard RGB cameras**: They can be found in smartphones and digital cameras. They capture images in RGB spectrum and are useful for general-purpose apple counting in well-lit conditions.
- **Infrared (IR) Cameras:** These sensors are useful for capturing images in low light conditions or for seeing through obstacles like plants with a lot of leaves.

Data acquisition involves how would the sensors collect data for the apple counting.

- **Placement and orientation**: Position camera at various angles and heights for having a clear view of the apples.
- **Lighting conditions**: Ensure adequate light in the background to create a vibrant image, help to distinguish between green apples and the leaves.
- **Field of View and Resolution:** Adjust the field of view (FOV) to capture the required area. Having higher resolution ensures apples are much visible and can be distinctively counted without any data overheads.

The datasets utilized in both approaches for this project were obtained from the Data Repository of the University of Minnesota, specifically from the work titled "MinneApple: A Benchmark Dataset for Apple Detection and Segmentation" by Haeni et al. (2019). The dataset can be accessed through the following link: https://conservancy.umn.edu/handle/11299/206575.

The images captured in this dataset were taken on a bright sunny day ensuring overall decent quality. The red apples are marked properly but some of the green apples were overexposed to bright sunlight. The trees' leaves were dark green, and the apples were light green in colour which helped in differentiating the green apples from the red ones. There was a substantial amount of variation in the dataset due to differing lighting conditions like degrees of shade and clustering of apples. However, the variation is within an acceptable range and thus the dataset is appropriate to be used in this project.

These were grayscale mask images in the dataset with a black background and grey colour highlighting the position of the apples. A Python script was written to load the images and corresponding mask, identify individual apples in each image and return their bounding boxes, which were used in object detection in later stages.

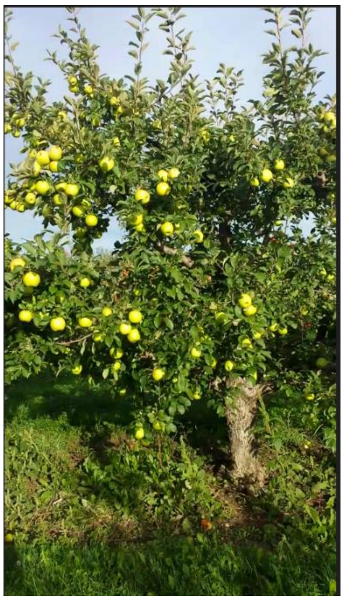


Figure 1 Image

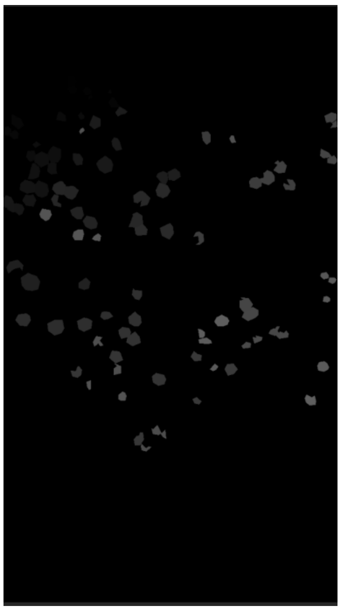


Figure 2 Mask

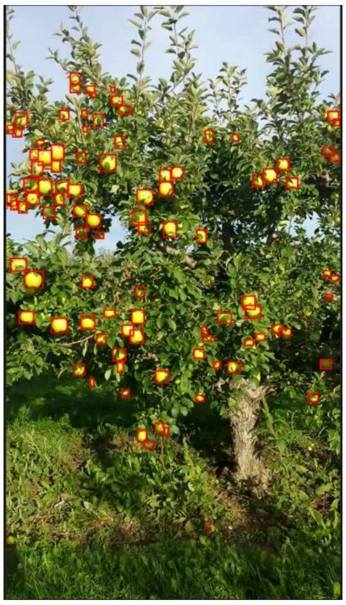


Figure 3 Bounding Box Image

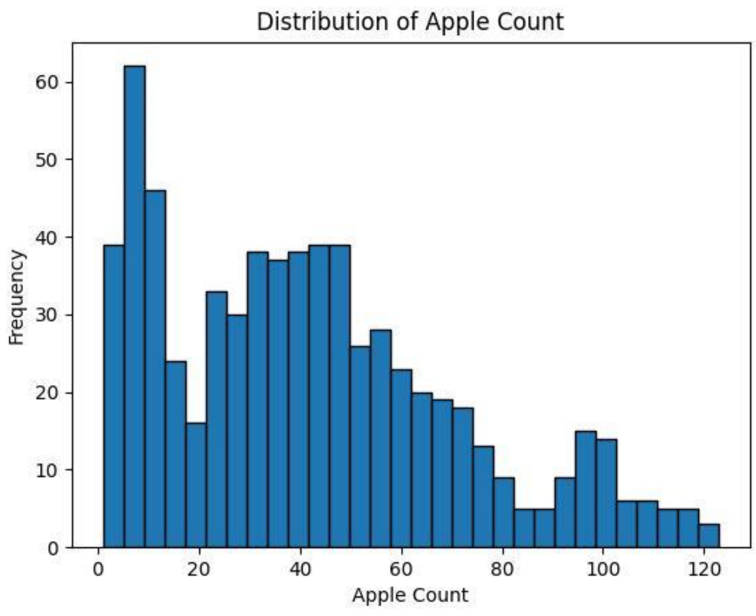


Figure 4 Number of images vs Number of apples

Methodology

A. Conventional Method

The methodology follows the principle of colour-based segmentation using the HSV colour space and contour analysis for feature extraction of red apples from an acquired image as an input. The main parameters for adjustment were Colour Range and Contour Area thresholds. In the Colour Range parameters, the lower and upper bounds of the HSV colour ranges were adjusted to identify red apples, whereas the contour area threshold was used to filter out small contours that might not correspond to actual apples. A step-by-step analysis of the approach is as below:

Image Acquisition and Pre-processing

The PNG images were loaded using OpenCV's **cv2.imread** function. Then they were converted from RGB to HSV colour space using **cv2.cvtColor**. This conversion simplified colour-based segmentation, making it easier to isolate green and red colours.

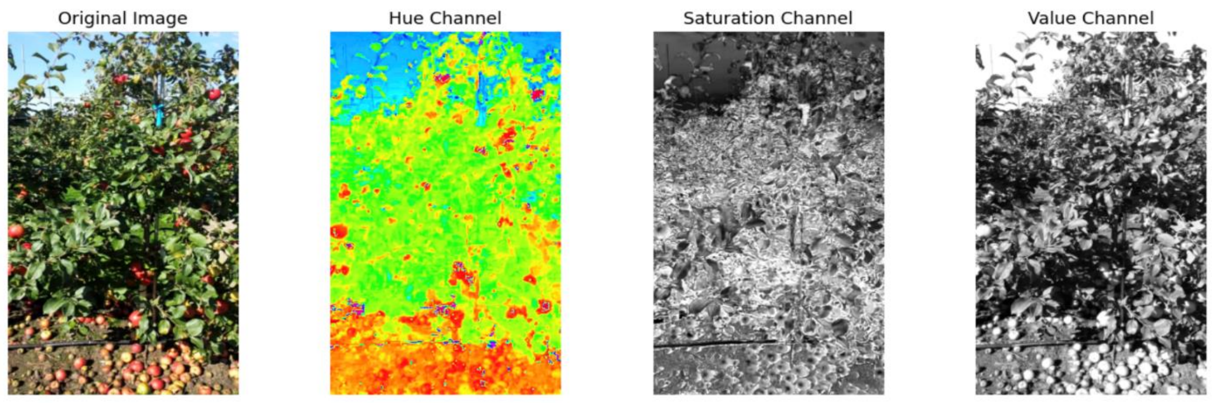


Figure 5 demonstrates the significance of hue channel in the case of red apples

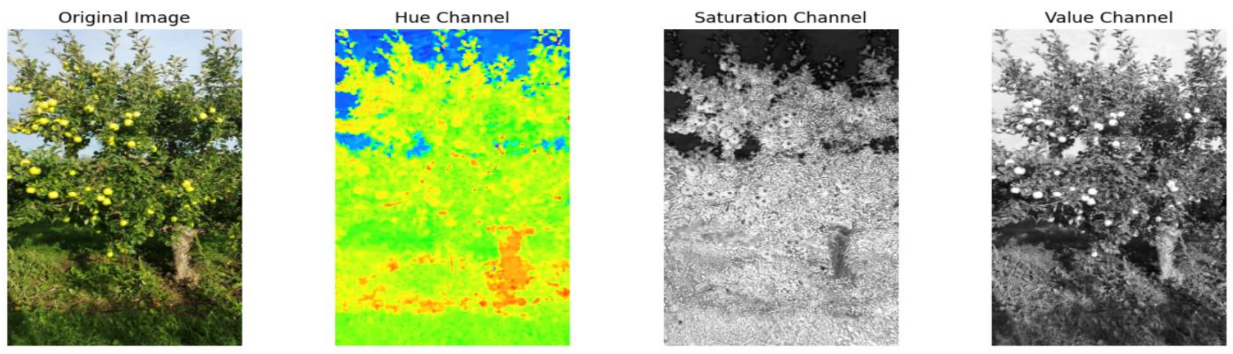


Figure 6 demonstrating the significance of value channel in the case of green apples.

The above figures clearly demonstrated the dissolution of the original image into its Hue, Saturation and Value components, and how the channels of the images can be tuned depending upon the colour of the apples to be identified. Finer and narrowed tuning would be required as the identification of green apples was more difficult.

Segmentation

Segmentation is the most critical part of the program. Colour-based segmentation was implemented to isolate the red pixels by creating a binary mask where the red pixels were assigned a value of 1. The lower and upper bounds for red colour in HSV were defined and adjusted based on various lighting conditions and shades of colour to identify red apples. This adjustment ensures that the segmentation process effectively captures desired red colour range.

Upon creating a threshold range for the red colour, a mask was implemented on the image segregating the areas that lie within the colour range.

1. Green Apple Detection

For identifying green apples, the code defines a lower and upper bound in the HSV colour space using **lower_green** and **upper_green** arrays. A mask is created using **cv2.inRange** to isolate pixels within the specified green colour range. Adjusting these thresholds is crucial for the accuracy of apple detection, and the values provided may need fine-tuning based on the characteristics of the input images.

2. Red Apple Detection

Similarly, the code defines thresholds for red apple detection using **lower_red** and **upper_red**. This process is essential in distinguishing red apples from the background. Like the green thresholds, these values might require adjustment for optimal performance across various images.

Feature Extraction

Contours in the created masks are identified using **cv2.findContours.** The contours were applied on the original image such that the number of apples in the image could be counted. The contoured were iterated through and then computed the area of each contour with a threshold value of pixels which was used to attenuate noise.

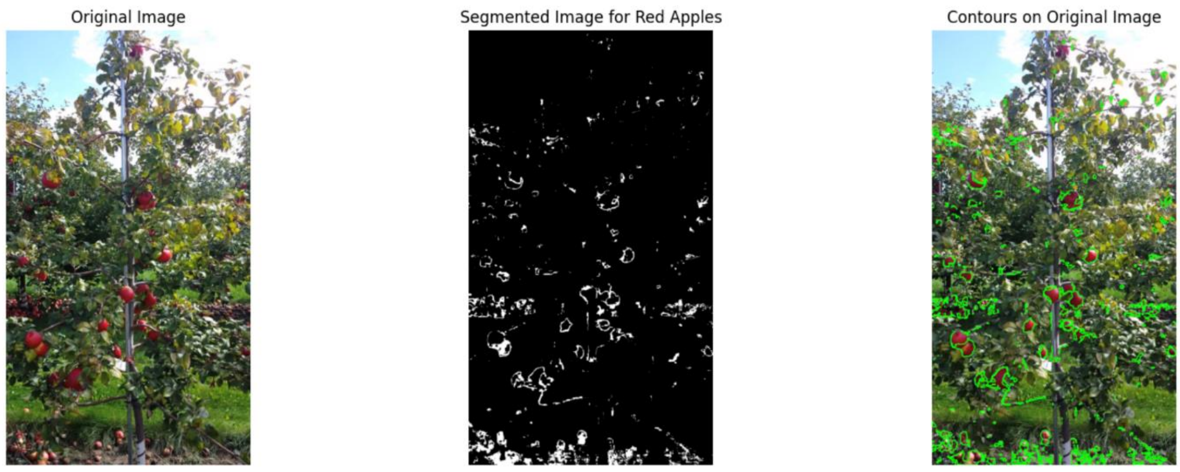


Figure 7 Segmentation result for red apples

The above figure shows the result of segmentation of red apples with lower bound value and an upper bound values adjusted with a saturation threshold of 50. Contours were drawn on the segmented image which attempted to identify the red apples in the image.

Visualization

The code employs **matplotlib** to visualize the HSV image before applying the colour mask. This aids in understanding the colour representation and the effect of the chosen thresholds. Additionally, the result with contours drawn around detected apples is displayed using **plt.imshow**. The Algorithm (pseudocode) used is demonstrated in the next section.

Apple Counting

After detecting apples, the code proceeds to count and visualize the results. Contours are drawn on the original image using **cv2.drawContours**, and the total count is displayed using **cv2.putText**.

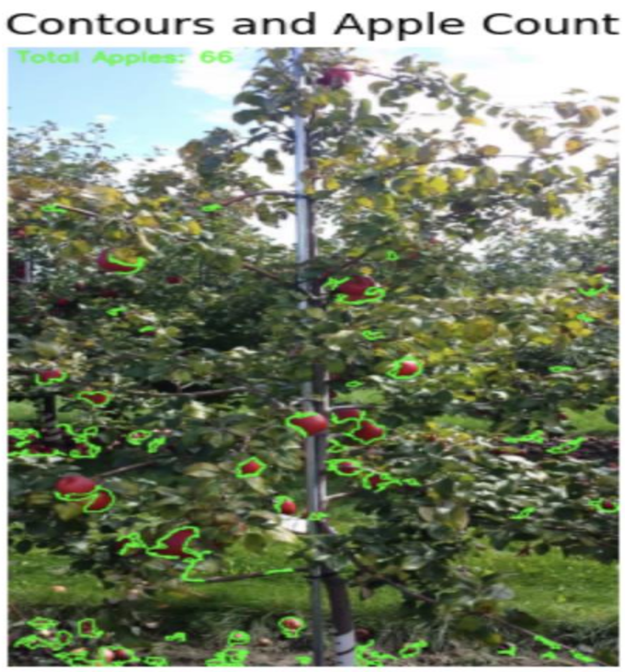


Figure 8 demonstrating the counting and visualization as the final outcome

```
      function identify_red_apples(image_path):
       apple <-- read_image(image_path)
# Read the image
        hsv_image <-- convert_to_hsv(apple)
# Convert the image to HSV color space
        lower_red <-- [0, 90, 90]
 # Define lower and upper bounds for red color in HSV
        upper_red <--[20, 255, 255]
       red_mask <-- create_red_mask(hsv_image, lower_red, upper_red)
 # Create a mask for red pixels (here it is for the red apples,
 it equally works for green apples in a different threshold)
       contours <-- find_contours(red_mask)
# Find contours in the mask
       count <-- 0
     for contour in contours:
# Draw contours on the original image and count the apples
        area <-- calculate_contour_area(contour)
        if area > 50:
# Adjust the threshold for area based on your image
            draw_contour(apple, contour)
            count <-- count+1
# counting of contours in the image
# output
display_apple_count(apple,count)
```

Algorithm 1 Conventional Machine Vision

B. Machine Learning Method

In the literature review, there have been past works that aimed to count apples or fruits in orchards by first detecting the fruits and then count the apples in the bounding boxes. One of the drawbacks of these approaches is that the CNN being used (e.g., YOLOv5) was primarily trained to perform object detection of small number of objects in an image that was not converged. As [7] shows, as the number of objects in the image goes over 100, a regression-based approach becomes several times more accurate in counting objects. Therefore, the apple counting task can be treated as an end-to-end regression problem which eliminate the estimation of bounding boxes.

Our approach involves building a CNN architecture with reference to YOLOv2 with additional dense layers at the end. The single output neuron estimates the number of apples in the image. The idea behind this approach is to allow the CNN to automatically learn an implicit representation of the number of apples instead of performing detection and counting separately.

The YOLOv2 blocks were chosen because the YOLO architecture is also designed to perform regression although it outputs bounding box coordinates instead of object counts.

The main structure comprises of blocks consisting of:

1. Batch Normalisation
2. Conv2D layers with Batch Normalisation and Leaky ReLU activation function
3. Max Pooling Layer

Following this, there are densely connected layers with leaky ReLU activation and the final output neuron, which gives the count of apples in the image, has a linear activation function.
A single block of our CNN structure is depicted below:

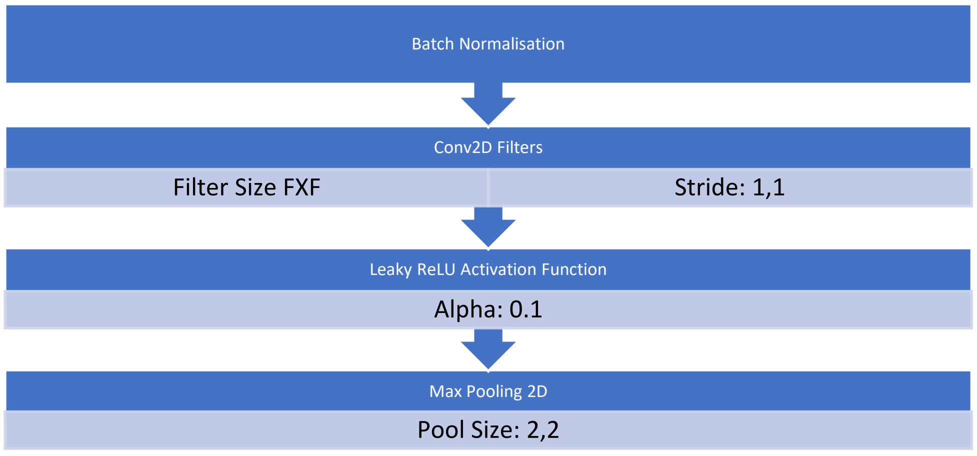


Figure 9 Iteration process in CNN

Block Number	Layer	Number of Filters	Size	Stride
Block 1	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	32	5x5	1,1
	Max Pooling 2D	1	2,2	-
Block 2	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	64	5x5	1,1
	Max Pooling 2D	1	2,2	-
Block 3	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	128	5x5	1,1
	Max Pooling 2D	1	2,2	-
Block 4	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	64	1x1	1,1
	Max Pooling 2D	1	2,2	-
Block 5	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	128	5x5	1,1
Block 6	Batch Normalization	1	-	-
	Conv2D (Leaky ReLU)	256	5x5	1,1
	Max Pooling 2D	1	2,2	-
	Global Average Pooling			
Block 7	Dense Layer (Leaky ReLU)	1024 Neurons		
	Dense Layer (Leaky ReLU)	128 Neurons		
	Output Layer (Linear Activation)	1 Neuron		

Table 1 Full Structure of the Apple Counting CNN

The Conv2D filters have a constant size of (3, 3) and a stride (1, 1), with the number of filters increasing in the layers closer to output in order to detect higher level features such as apples in a bunch.

Since the task consists of counting apples, it is expected the feature detection aspects of the CNN (such as edges and other low-level features) are not as complex as a general object detector. Furthermore, it is observed to be overfitting due to CNN. Therefore, we have

1. Structured the network to be smaller than YOLOv2 by removing the Convolutional layers with the most number of filters (512 and 1024).
2. Replaced some *{[FxF] – [1x1] – [FxF]}* blocks with just *[FxF]* bocks where *F* is the filter size.
3. Added an additional dense layer at the end to improve regression performance.

Features of the CNN are discussed below:

Batch Normalisation:

For each mini batch, the inputs to the layer is normalised by calculating the mean and standard deviation. This provides a regularisation effect and helps the training to converge faster.

Leaky ReLU activation function:

This activation function is similar to ReLU activation except for a slight negative slope when <0.

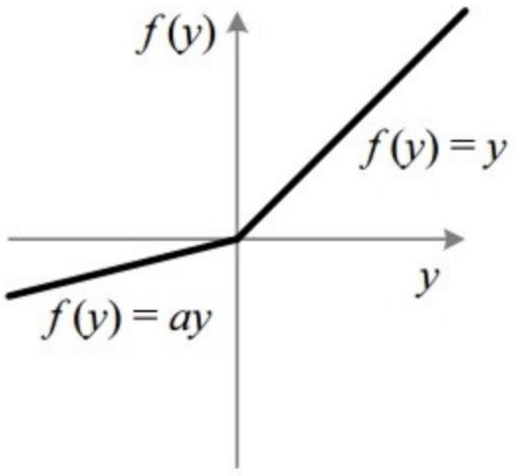


Figure 10 Leaky ReLU Activation function[8]

The activation function adds nonlinearity to the CNN to enable learning of complex features. The slopes of the Leaky ReLU function reduce the effect of vanishing gradients during learning and results in faster learning.

Adam Optimiser [9]:

The Adam (Adaptive Momentum) optimiser combines RMS prop and momentum-based optimisers to obtain advantages of both.

$$m_t = \beta_1 * m_t + (1 - \beta_1) * (\delta L / \delta w_t)$$
$$v_t = \beta_2 * v_t + (1 - \beta_2) * (\delta L / \delta w_t)^2$$
$$\hat{m_t} = m_t \div (1 - \beta_1^t)$$
$$\hat{v_t} = v_t \div (1 - \beta_2^t)$$
$$w_t = w(t-1) - \alpha * (\hat{m_t} / \sqrt{(\hat{v_t})} + e)$$

[10]

Figure 11 Adam optimizer equations where wt is the weight update

The Adam optimiser adjusts the step size to larger or smaller values according to the gradient resulting in a faster reduction in loss values and faster learning. It has been shown to outperform other similar optimisers by a large margin [9]**.**

It has been used to perform training with the mean square error as the loss. The RMS error is used for validation after every epoch. The mini batch gradient algorithm descent has been utilised where the dataset is divided into smaller batches of images and labels, and the learning algorithm is performed for the batches successively.

Experiment and Implementation

Traditional Method

Algorithm 2 below illustrates the two key methods implemented: identify red and green apples in the entire dataset.

identify_apples function is developed which converts each image to HSV for enhanced detection. Images was loaded to colour picking tool and pixels values of the apples on the tree were selected for tracking. The range of these values were used to create a binary mask. The values are as below:

```
lower_value = np.array([0, 127, 214])
upper_value = np.array([255, 255, 255])
```

The array contains combinations of HSV values obtained from trial-and-error method.

As counting is dependent upon the pixel area a pixel area bound process is developed. On assiduously experimenting it has been found that an average value greater than 12-15 works very well.

Quantitative aspect involves calculating RMS error between detected and actual counts which measures detection accuracy. The ***plot_accuracy*** function plots the detected counts against actual counts, along with a line indicating perfect accuracy for visual presentation.

```
# Function to identify apples based on HSV color mask
Function identify_apples(image_path, lower_hsv, upper_hsv):
    Read and convert the image to HSV color space
        lower_red <-- [0, 90, 90]
        hsv_image <-- convert_to_hsv(apple)
# Define lower and upper bounds for red color in HSV
        upper_red <--[20, 255, 255]
       red_mask <-- create_red_mask(hsv_image, lower_red, upper_red)
# Find contours in the mask
     count <-- 0
     for contour in contours:
Count and return the number of apples based on contours

# Function to read the actual count from a text file
Function read_text_file(file_path):
    read, and return the content of the file

# Function to process all apple images in a directory
Function process_apples_in_directory(directory, color_hsv_bounds,
count_txt_directory):
    For each image in the directory:
        Detect apple count using identify_apples
    Return lists of detected and actual counts

# Function to calculate RMS error between detected and actual counts
   calculate_rms(detected_counts, actual_counts):

# Function to plot detected counts against actual counts
Function plot_accuracy(actual_counts, detected_counts, title, color):
Plot a scatter plot and a line of perfect accuracy
Set labels, title, and grid for the plot

# Define paths and HSV bounds for green and red apples
green_hsv_bounds = (lower_green_hsv, upper_green_hsv)
red_hsv_bounds = (lower_red_hsv, upper_red_hsv)
# Process Green Apples: Detect, calculate RMS error, and plot accuracy
green_detected_counts, green_actual_counts = process_apples_in_directory
green_rms_error = calculate_rms
plot_accuracy(green_actual_counts, green_detected_counts, 'Green Apples
Accuracy', 'green')

# Process Red Apples: Detect, calculate RMS error, and plot accuracy
red_detected_counts, red_actual_counts = process_apples_in_directory
red_rms_error = calculate_rms
plot_accuracy(red_actual_counts, red_detected_counts, 'Red Apples Accuracy',
'red')
```

Algorithm 2 Functions used in apples detection.

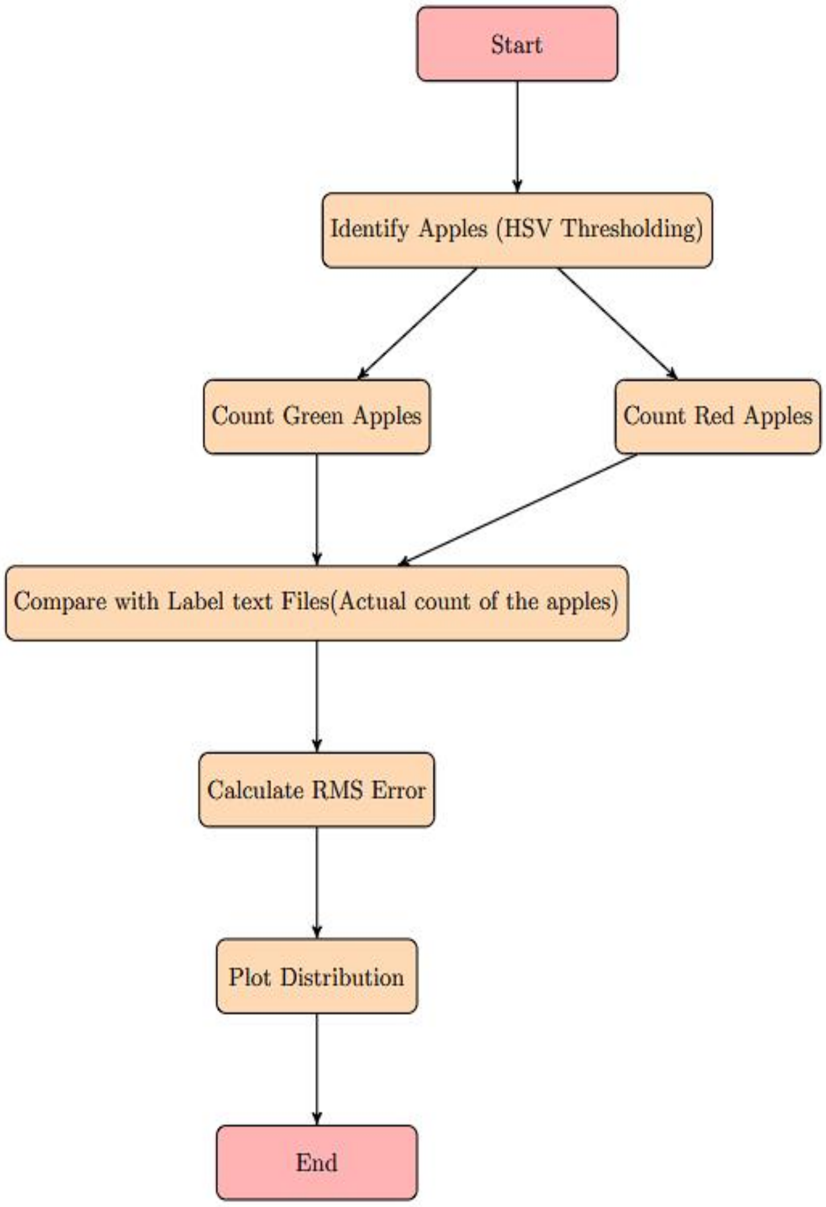

Start
Identify Apples (HSV Thresholding)
Count Green Apples
Count Red Apples
Compare with Label text Files(Actual count of the apples)
Calculate RMS Error
Plot Distribution
End

Figure 12 Flow chart for traditional method

Machine Learning Method

The CNN architecture was developed using Keras and trained on Google Colab. Keras is an API for the popular Machine Learning framework *TensorFlow*. Other Python packages used include Numpy for numerical operations and OpenCV for image processing. The input images were resized down to *(640x360),* and the pixel values scaled between 0 and 1. The images and ground truth values were shuffled and converted to Numpy arrays for training. The Batch Normalisation Layers in the CNN is expected to account for the normalisation operation that may be required on the input data.

The CNN was trained on T4 GPU on Google Colab. Some of the hyperparameters that were adjusted include batch size, filter size, image size, and the optimizer used for training. The dataset was split into train, validation, and test in the ratio (0.7: 0.1: 0.2). The "mean squared loss" was used as the loss function and "root mean squared error" as the metric on the validation set. The mean squared error computes the error between the number of apples estimated by the CNN vs the ground truth.

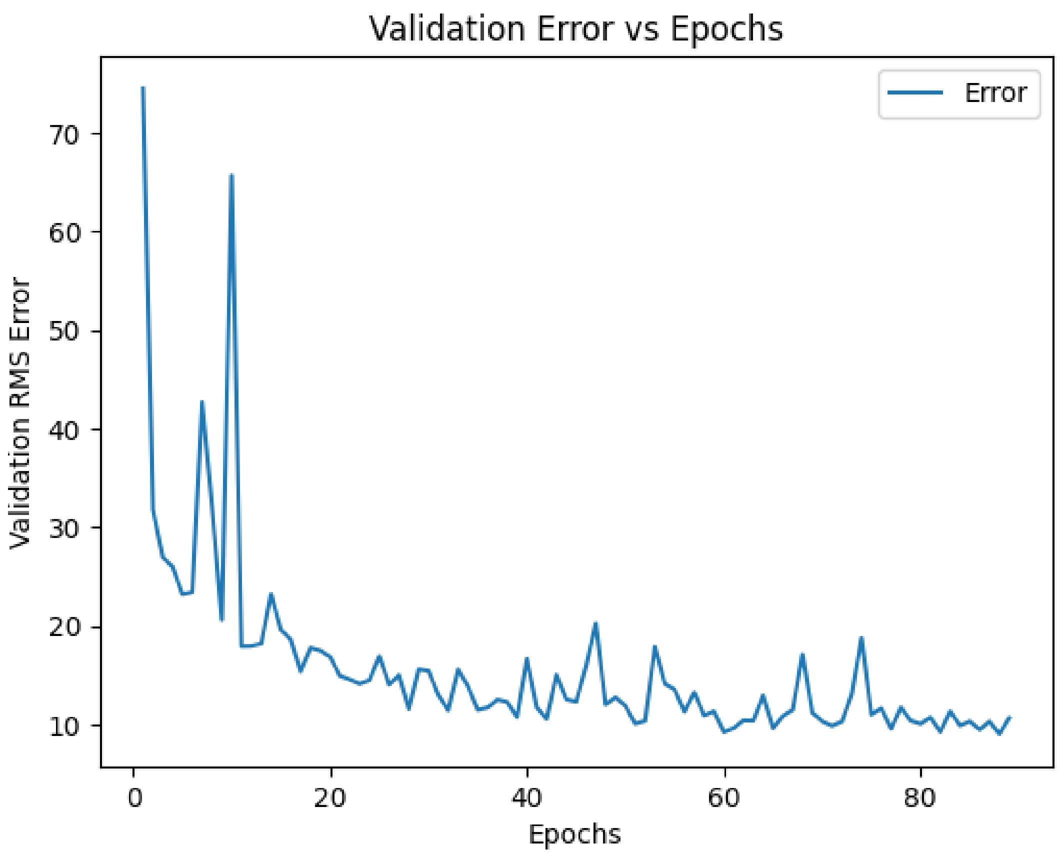


Figure 13 Validation Error vs Epochs during training

We observed that the batch size parameter had a significant effect on the training performance. The CNN tended to overfit into larger batch sizes (16 and above) leading to poorer performance on validation sets. By setting the batch size to 8, this issue was mitigated as the smaller batch size had a regularising effect and lowered the overfitting.

The training was done for 100 epochs as the upper limit. The training was continued for many additional epochs in order to ensure that a global minima was reached as, the stochastic nature of the learning process caused the loss function increase at times. Once a stable validation loss was achieved, the training was stopped. We ascertained these epoch limits to be fair from experimental observations in the trends of learning and time constraints while training. The weights of the CNN were saved every 5 epochs and the weights corresponding to the lowest validation loss values were used to obtain the results.

Results and Evaluation

Traditional Method

When using HSV colour technique for apple detection, the main challenges are as follow:

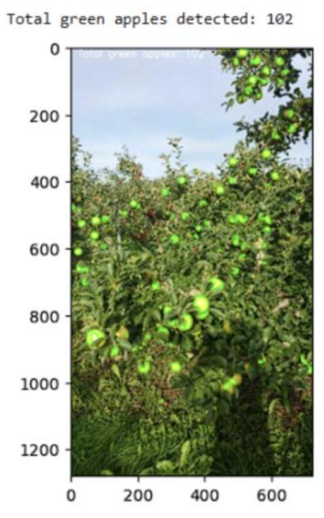
1. Variation in colour and shades make defining a fixed robust HSV range for all apples a difficult task as it makes appropriate thresholds selection challenging. It is particularly more difficult for the detection of green apples as a narrow range misses many green apples and a broad range causes even the dark green leaves of an apple tree to be mis-read as a light green apple.

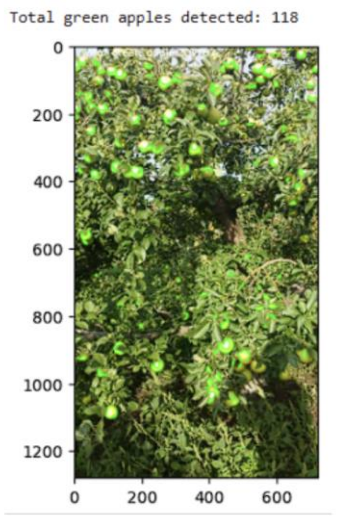
2. Changing illumination setting also has major implications on the chosen HSV range which may need adjustment to accommodate variations in illumination.

3. The size and orientation of the apples in the image can make an impact on the contour-based detection's performance. Small or uneven apples may be difficult to be recognized. This is also the case where the apples are further away from the camera and smaller than the minimum area defined for the contour.

4. If the dataset has bad photo capture, then identifying apples in a single run over the entire dataset become exceedingly difficult. In such cases, only one image at a time, with values adjusted can be used defeating the entire purpose of using this method.

Some Results using Traditional methods in Machine Vision (HSV colour technique):





Determined counts of Apples

Figure 14 and 15 Apple count

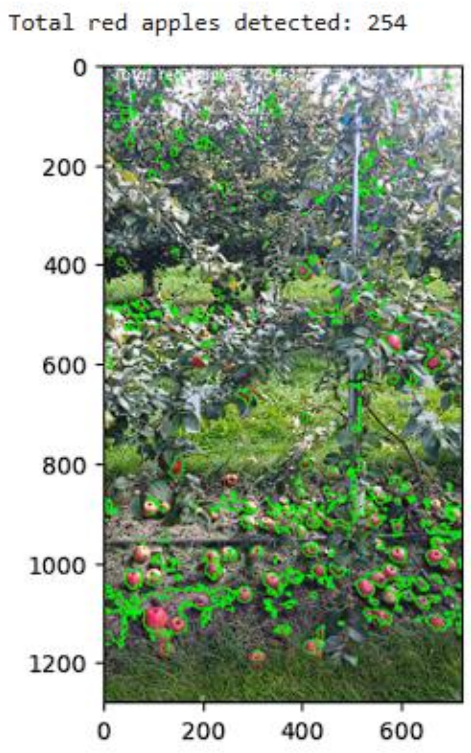


Figure 16 Red_apple_1

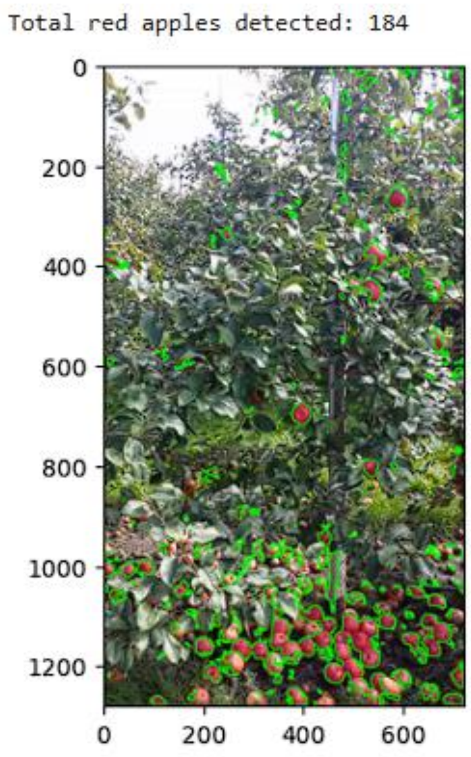


Figure 17 Red_apple_2

Calculating Performance

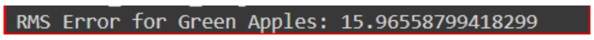
To calculate error, 100 data points were manually selected containing green and red apples respectively and were stored in two separate files. Then the apple counting algorithm was applied to both folders. Each image has an associated ".txt" file, which provides the actual count of apples in that image. With this value and the ground truth value, RMS error is calculated. A challenge faced in getting the RMS was that it was hard to detect some apples due to changing lighting conditions thus a filter was added:

```
apple_count = 0
   for contour in contours:
       area = cv2.contourArea(contour)
       if area > 25 # keep around 10-30 for better detection
       apple_count += 1
```

Algorithm 3 Filter

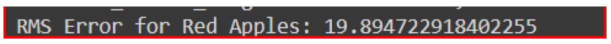
This code segment counts the number of apples in an image by iterating through detected contours (the outlines of objects). For each contour, it calculates the area and counts it as an apple if the area is larger than a specified threshold (25 in this case). This process helps differentiate actual apples from smaller, irrelevant objects or noise in the image.

$$\boldsymbol{RMS} = \sqrt{\frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}}(\boldsymbol{predicted_i} - \boldsymbol{actual_i})^{\mathbf{2}}}$$



```
RMS Error for Green Apples: 15.96558799418299
```

Figure 18 RMS Error for Green Apples



```
RMS Error for Red Apples: 19.894722918402255
```

Figure 19 RMS Error for Red Apples

Machine Learning Method

The metrics include RMS error and percentage error between detection result by the CNN and ground truth data were computed as follows.

The apple count error is measured as

$$\frac{Detected\ Apples - \ Actual\ Apples}{Actual\ Apples} * \ 100$$

The metrics were measured using conventional method and the ML method for

1. dataset containing the red apples
2. dataset containing the green apples

After training for 90 epochs, the ML method performance was measured on the test dataset, which was not used in the training process.

RED APPLE DATASET RMS ERROR = 5.54437

RED APPLE DATASET COUNT ERROR: 0.25%

GREEN APPLE DATASET RMS ERROR = 7.1232

GREEN APPLE DATASET COUNT ERROR: -0.58%

TESTING DATASET COUNT RMS ERROR: 10.2105

TESTING DATASET APPLE COUNT ERROR: 1.2743%

The metrics describe that the error in detecting red apples is lower than that of green apples. It is justifiable since red apples are usually larger relative to the image size (due to lower count in the image) thus easier to identify in the images. The similar colouration between green apples and the trees in the background also make the green apples more difficult to be identified.

The percentage error of counting red and green apples respectively are very close or less than 1%, and this is the case as applied to the overall testing set error as well. However, the RMS error is observed to be in reasonable ranges for the red apple dataset, the green apple dataset, and the testing dataset.

Though all the results were computed using weights after 90 epochs of training, it should be noted that RMS and percentage error on test dataset after the CNN was trained for 45 epochs was found to be 15.3725 and 6.066%. From 45 to 90 epochs the validation error saw a slight decrease with fluctuations and the test dataset RMS error decreased by 5.162. There is possibility of an overfitting behaviour of the CNN model. Further testing needs to be done and if required the hyperparameters can still be refined or more effective regularization techniques can be implemented into training.

Comparing Conventional and Machine Learning Method

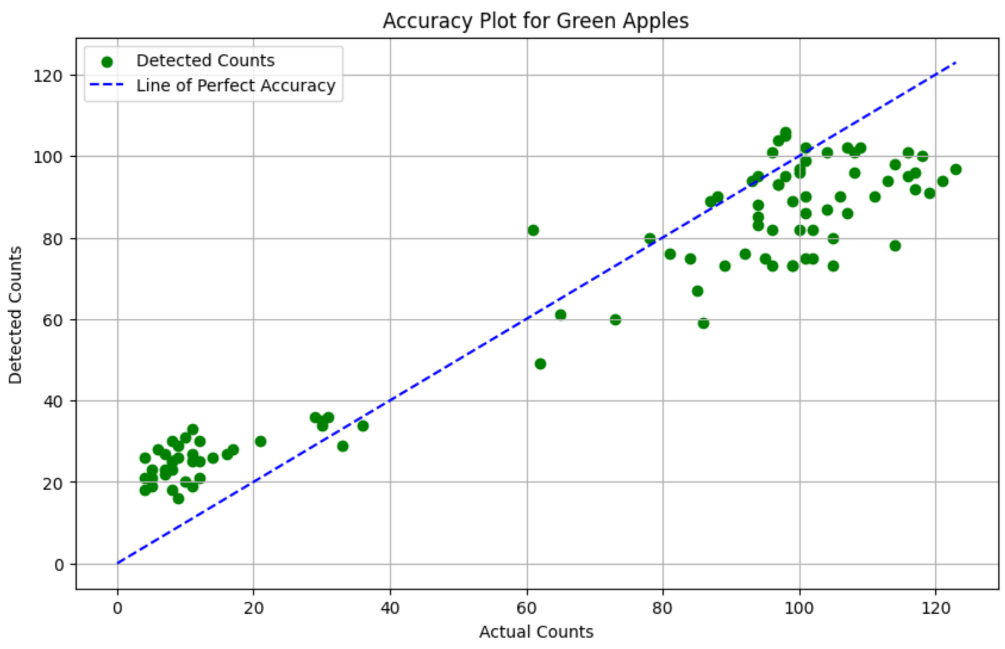


Figure 20 Accuracy plot for green apples Traditional method

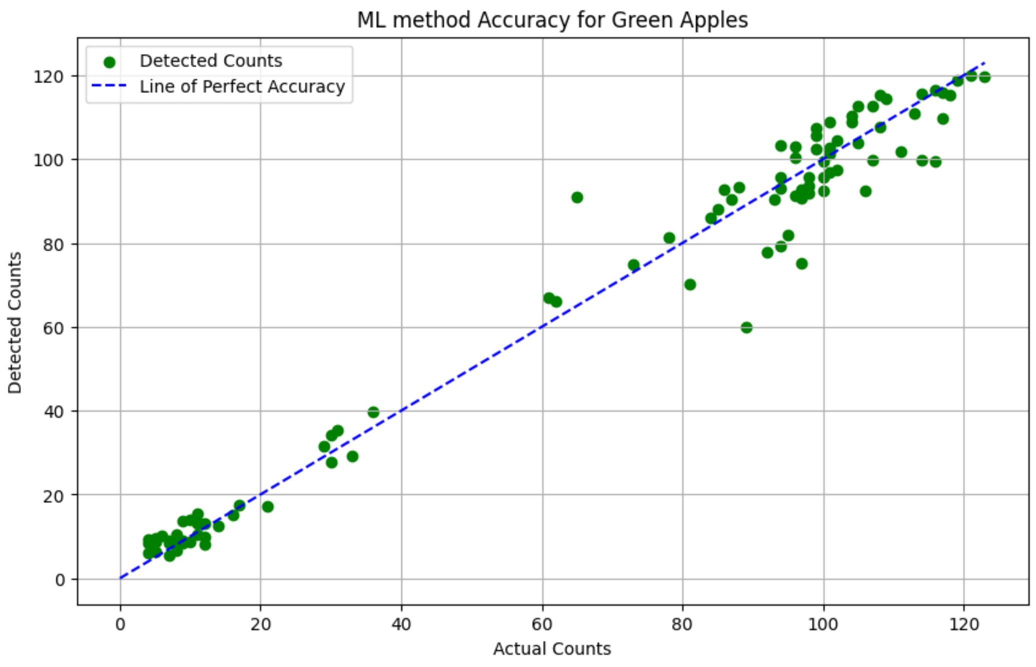


Figure 21 Accuracy for Green apples ML method

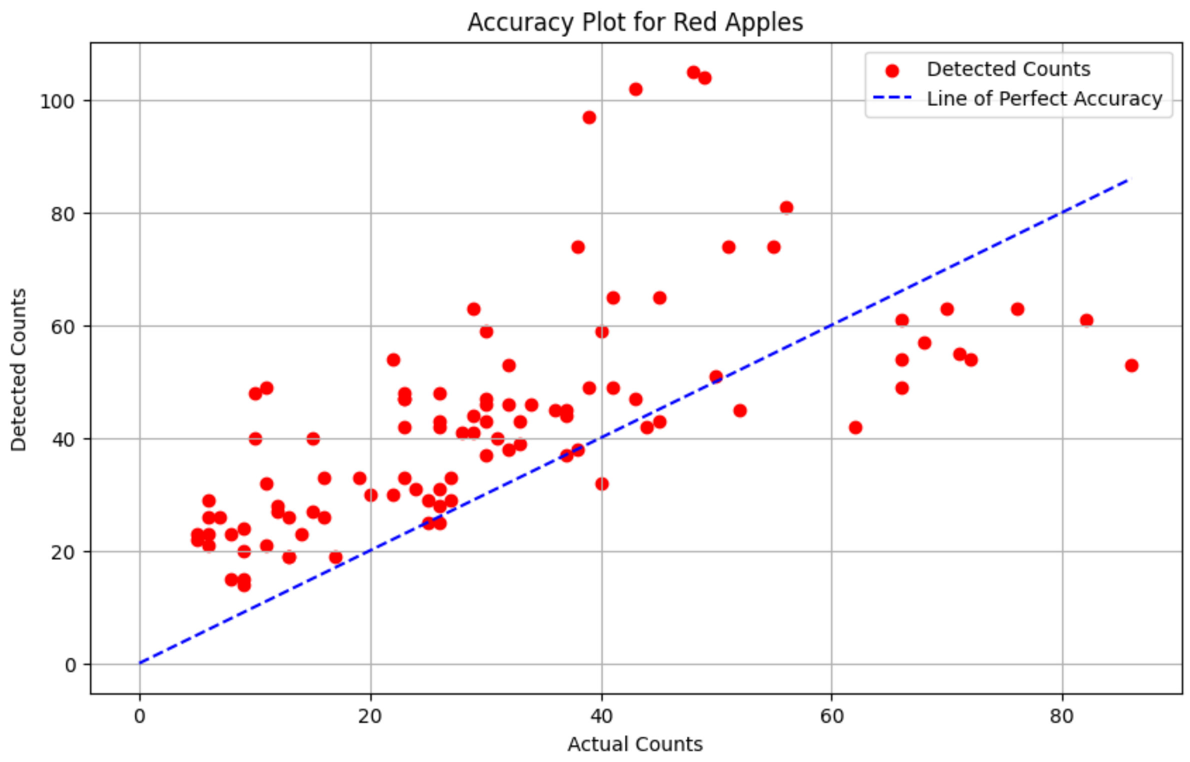


Figure 22 Accuracy plot for red apples Traditional method

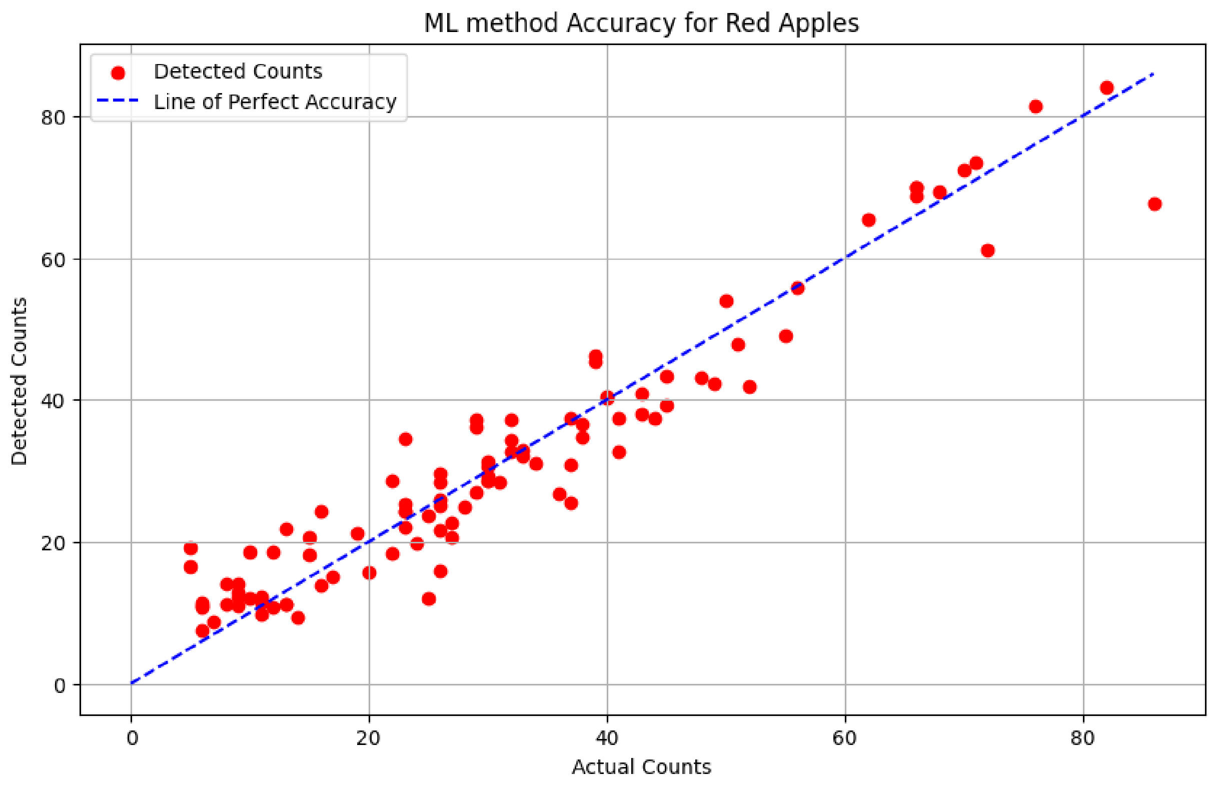


Figure 23 Accuracy for red apples ML method

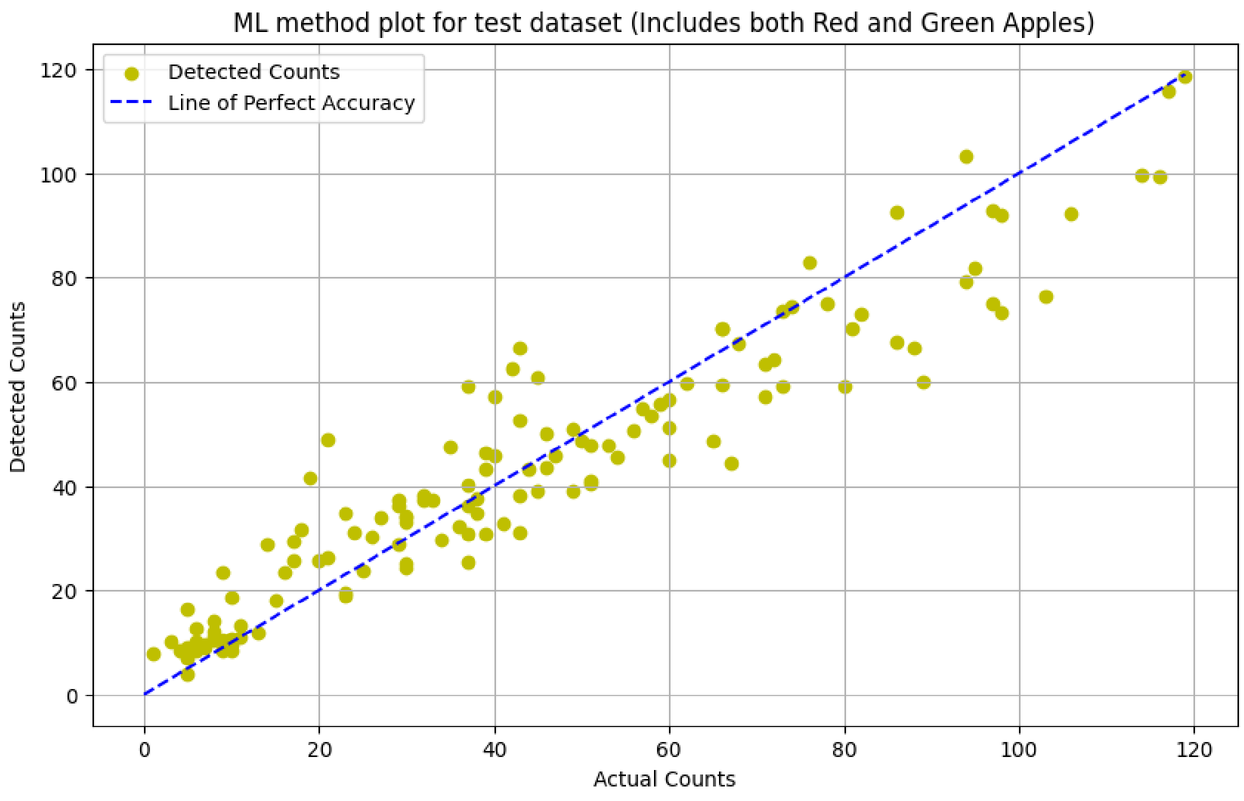


Figure 24 Red and Green apples detection ML method

The plot visually compares the detected counts of apples (as determined by our algorithm) against the actual counts (as recorded in the ".txt" files). Here's what the plot represents and how it illustrates accuracy:

- Points close to the blue line indicate high accuracy, as the detected counts are close to the actual counts.
- Points further away from the blue line indicate lower accuracy, as there's a larger discrepancy between the detected and actual counts.

From the graphs above, the detected apple counts of traditional machine vision method are apparently disperse, especially for red apples. There are only a handful of counts that lie on the accuracy line. These points are relatively away from the line which indicates a higher discrepancy. Furthermore, the general trend with the red apples dataset observed where the detected count is higher than the actual count can be attributed to the method erroneously detecting apples on the ground and duplicate detections.

On the other hand, the detected apple counts of machine learning method are much more converged that a trend can be seen from the plot. The accuracy is very close to perfect when the number of apples in an image is less than 20. There are more points lying on or very close to the accuracy line which indicates a better detection performance of using a CNN model over traditional image processing method.

Another reason why machine learning approach is better than the traditional method is that it can detect any coloured apple no matter the red or green colour it will count it. However, for the traditional method the dataset has to be separated based on red and green apples due to different colour segmentation.

Conclusion and Future Works

Apple counting based on traditional machine vision and ML method were done and their performances were demonstrated with a realistic dataset. The limitation of both methods is that it is unlikely to detect all apples in an orchard correctly without missing or false detection due to vast amount, camouflage effect between the object and the background, and object pixel size.

Completing such tasks with conventional image processing method has low initial latency and less processing power consumption than using Machine Learning method. However, the experiment proved that using ML method provides a higher accuracy over image processing method in solving these tasks.

Future work can include the use of k-fold cross validation in testing and evaluation the overfitting issue with ML method. More experiments can also be done with new and different datasets. Besides, the data capture can be held in video format so that the model can also compare detection results within frames to eliminate missing and false apple detections.

In hardware perspective, if the collection of images can be held with drones, the image capturing will be in a top-down view. In such case, it is hypothesized that the accuracy of the detection will increase.

References

[1] "A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5," 21 April 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/9/1619. [Accessed 2024 January 6].

[2] "A Two-Stage Deep-Learning Model for Detection and Occlusion-Based Classification of Kashmiri Orchard Apples for Robotic Harvesting - Journal of Biosystems Engineering," 06 june 2024. [Online]. Available: https://link.springer.com/article/10.1007/s42853-023-00190-0. [Accessed 6 january 2024].

[3] "Apple Detection in Complex Scene Using the Improved YOLOv4 Model," 04 march 2021. [Online]. Available: https://www.mdpi.com/2073-4395/11/3/476. [Accessed 06 janaury 2024].

[4] international journal for research in applied science and engineering, 12 December 2021. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.ijraset.com/best-journal/object-tracking-using-hsv-values-and-opencv. [Accessed 5 january 2024].

[5] "Automatic recognition vision system guided for apple harvesting robot," 30 november 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790611001819. [Accessed 08 january 2024].

[6] G.-Q. J. a. C.-J. Zhao, "Apple recognition based on machine vision," 2012 International Conference on Machine Learning and Cybernetics," Xi'an, China, 2012.

[7] A. S. Gomez, E. Aptoula, S. Parsons and P. Bosilj, "Deep Regression Versus Detection for Counting in Robotic," IEEE Xplore, 2021.

[8] "Papers with Code - Leaky ReLU Explained," [Online]. Available: https://paperswithcode.com/method/leaky-relu.

[9] "Adam: A Method for Stochastic Optimization," arXiv.org, 30 janaury 2017. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed 09 janaury 2024].

[10] "Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki," Adam, [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=Adam. [Accessed 8 January 2024].