APPLE COUNTING IN AN ORCHARD

By

Lohit Yarra, (22071774)

Sathvik Kadimisetty, (22071773)

Adip Ranjan Das, (22071780)

Atharva Kulkarni, (22071782)

Sakshi Jadav, (22071797)

UFMFRR-15-M Machine Vision Report

Department of Engineering Mathematics

UNIVERSITY OF BRISTOL

&

Department of Engineering Design and Mathematics

UNIVERSITY OF THE WEST OF ENGLAND

January 9, 2023

Contents

List of Tables

1 Introduction

Automated apple counting is crucial in agriculture because it allows for accurate forecasting, inventory management, and yield predictions. Deep learning is a type of machine learning in which data is processed and analysed using multiple layers for neural networks. Large datasets, which are required for object detection, can be used to train these models. Once trained, the model can learn to recognise a diverse set of images and a wide range of apples while considering various aspects of the possible environment. While manual counting can be time-consuming and error-prone, image processing and machine learning techniques provide promising alternatives. We proposed, implemented, and compared a traditional image-processing and machine-learning approach for apple counting in this report. We aim to evaluate these methods' accuracy, efficiency, and robustness thoroughly. We also offer insights into each approach's strengths and limitations, aiming to inform future research and development in this field.

2 Related Works

2.1 History Before Deep Learning

2.1.1 Feature-based object detection

Using distinguishing traits that may be extracted from photos, feature-based object detection attempts to identify objects. The scale-invariant feature transform (SIFT), created by David Lowe in 1999, is one of the most popular feature-based object identification techniques.

2.1.2 Template Matching

This straightforward approach of object detection includes matching a small image (template) with parts of a bigger image to find a match. A notable contribution to the development of template matching could be The development of the Normalized Cross Correlation (NCC) measure, which is a common metric used to evaluate the similarity between the template and the image.

2.1.3 Edge-based object detection

Identifying items in an image by locating their edges is known as edge-based object detection. The Canny edge detector, created by John Canny in 1986, is one such edge-based object detection technique. The basis of the Canny edge detector is the concept of identifying "strong" edges (edges with high contrast) in an image and connecting them to generate a contour. This algorithm uses 4 different filters to identify the edges in different orientations.

Edge detection in HSV color space

The development of algorithms like the Color-Based Probabilistic Tracking algorithm and the HSV (Hue, Saturation, Value) Color Model for Object Detection allowed for the application of the HSV

colour space in the field of object detection as early as the early 2000s. These methods examined the hue, saturation, and value channels independently in the HSV colour space to identify objects based on their colour. The HSV color space can be used in conjunction with edge-based object detection to improve the accuracy of the object detection particularly in applications where the color and shape of the object are important features.

2.2 The Deep Learning Era

In the beginning stages,taking into account objects with "different sizes" and "different aspect ratios" was necessary for multi-scale object recognition this was one of the major technological difficulties of object detection. Algorithms that can run effectively on specialised hardware including graphics processing units are used to enable real-time object recognition (GPUs). Convolutional Neural Networks (CNNs): CNNs are a type of neural network designed specifically for image recognition tasks. They are commonly used in object detection models because they are able to learn and extract useful features from images automatically, without the need for manual feature engineering. The two categories of detection networks are based on the candidate regions (two-stage detector) and the regression approach (one-stage detector) Region-Based Convolutional Neural Networks (R-CNNs): R-CNNs are a type of object detection model that uses CNNs to extract features from a region of interest in an image, and then uses a separate classifier to classify the object and predict its location.

Fast R-CNNs: Fast R-CNNs are an improvement on the R-CNN model that uses a shared convolutional layer to process the entire image, rather than individual regions. This allows the model to be more efficient and faster to train. Girshick in in April 2015 presented the selective search approach to improve the model's object detection accuracy called Fast RCNN.

Faster R-CNNs: Faster R-CNNs are another variant of the R-CNN model that uses a region proposal network to generate region proposals, rather than using a sliding window approach. This allows the model to be more efficient and accurate.

You Only Look Once (YOLO): YOLO is a single-shot object detection model that uses a fully convolutional network to predict the bounding boxes and class probabilities for objects in an image. YOLO is known for its speed and efficiency, making it well-suited for real-time applications.

Single Shot Multibox Detector (SSD): SSD is another single-shot object detection model that uses a CNN to predict bounding boxes and class probabilities for objects in an image. SSD is known for its speed and ability to handle objects of various sizes.

2.3 Lohit Literature Review

Here we look into some of the methods that were researched to understand previous work.

Author	Type of Fruit	Methodology	Metrics
Guantao et al.(2020)	Apples	Faster R-CNN, Improved YoloV3	Training loss curve and mAP score curve
Steven W. Chen et al.(2017)	Apples and Oranges	Blob detection, Count and linear regression	lsquare error, Ratio Counted, STD Dev
Leonardo et al.(2020)	Apples	CNN, Object detection	Average precision, true-positive, false-positive, Intersection over union

Table 2.1: Papers reviewed

Here as we have a table mentioning the papers that were taken for understanding and further exploring the current advancements.

The first paper by Gauantao et al.(2020)[1] had different detection models studies and compared for different interference conditions. The models are AlexNet + Faster RCNN, ResNet101+Faster RCNN, DarkNet53+ Yolov3, Improved Yolov3. Transfer learning was used to accelerate the training process. Results here showed that the improved YOLOV3 model had the best recognition effect amongst them. It also had better performance for occlusion, spot, overlap and incomplete apples with a recognition recall higher than 88.5

The second paper by Steve E. Chen wt al.(2020)[2] used Blob Detection neural network, count neural network and final count linear regression. The performance of count for the apples and oranges were used declared and shown by showcasing l^2 error, ratio counted and STD Dev. Also, the performance for distinct datasets of orange in daylight and green apples at night, utilizing human generated labels as ground truth. The l^2 error was best in case oranges as compared to apples. There is a limitation here human generated labels are prone to error which are not desirable. The positive aspect here would be the accurate counting from the limited dataset and short labelling time.

The third paper by Leonardo et al. (2020)[3] used a ATSS deep learning based approach. Here initially, other object detection methods were first studied namely RetinaNet, R-CNN, Cascade

and R-CNN, Faster R-CNN, Feature Selective Anchor Free and HRNet. Then the main proposed method Adaptive Training Sample Selection was proposed which are operated on close range and low cost terrestrial RGB images. Here, the main advantage of the ATSS method is that only the centre point of the objects is labelled, which worked better than relying on bounding box annotations in heavily dense fruits.

For Deep learning approach, we decide on YOLOv5 after much deliberation, thought the ATSS method provides better solution for apple counting that YOLO. This is due to the significant difference in the dataset we are using. The ATSS dataset consists of large number of clustered apples, whereas our are very minimal in comparison. Yolo method was explored particularly in the first paper[1], which gave us the required data to adopt it for our project.

3 Data Acquisition and datasets

3.1 Data Acquisition

The images for this task were obtained from a dataset provided by the University of Minnesota (Häni et al., 2019). These images were obtained by video footage from different sections of the orchard using a standard Samsung Galaxy S4 cell phone. During data collection, video footage was acquired by facing the camera horizontally at a single side of a tree row. Individual images were extracted from these video sequences.

3.2 Datasets

The dataset consisted of three folders “Test data”, “Detection”, and “Counting”. This “Test data” folder consisted of counting, detection and segmentation, whereas the “Detection” folder consisted of images, JSON files mapping and ground truth, and the “Segmentation” folder consisted of images and masks.

We used the images in detection>train to build our yolov5 model, and this was run on the test images after validation with a small set of images in the training set was done. The 670 PNG files for training were uploaded to the cloud and synced with Google collab to run the dataset. Similarly, the test dataset was imported into the collab workspace file directory during the session from the cloud.

There are various types of apples and surroundings in the “train” dataset, for example, red apples, green apples, and a wide variety of distant and close views of apples, as well as differences in light and shadow in the environment, which complicates and makes detection considerably more difficult.

4 Research Methodology

In accordance with our literature review, we decided on using the combination of HSV masking and canny edge detection for detecting apples with the conventional approach, whereas for the deep learning approach, we went with YoloV5s. More about this is explained below.

4.1 Conventional Approach

The conventional approach in image processing refers to algorithms used to manipulate digital images to extract functional pieces of information. When applied to the image, some algorithms change the value of the individual pixel, whereas others use neighbouring pixel values to transform the individual pixel value. The conventional approach is helpful in image segmentation, adjusting the brightness and contrast, smoothing and sharpening the image and thresholding.

For the apple counting task, we used the open-source library OpenCV, a machine vision library and python programming language. Multiple filtering algorithms were used to extract the location of the apples in the image. We started by extracting the colour of the red apples using HSV colour space and creating a mask which detects the red colour. The mask was then passed through a blob detection algorithm that returned the apples' location and counts.

The SimpleBlobDetector is a machine-learning approach that detects blobs in an image. It is a fast and simple algorithm based on a Dense Feature Detector and uses Hu Moments to detect the shape of blobs. These blobs can be used in object tracking and counting.

"Image moments are the weighted average of image pixel intensities". It uses a set of 7 numbers to calculate the moments. The Hu Moments are invariant to translation, rotation or scaling.

4.2 Deep learning Approach (Using Yolo v5s)

YOLO (You Only Look Once) is a real-time object detection algorithm that divides the input image into a grid of cells and predicts the class and location of objects within each cell.

For the deep learning approach part of the experiment, we decided to use the YOLO v5 model for its ability to perform detection and classification in a single forward pass of the network, making it much faster than other object detection algorithms that require multiple passes or separate stages for detection and classification. Pre-trained checkpoints are as below.

Model	size$_{(pixels)}$	mAP$^{box}_{50-95}$	mAP$^{mask}_{50-95}$	Train time$_{300epochsA100}$	Speed$_{ONNXCPU(ms)}$	Speed$_{TRTA100(ms)}$	params$_{(M)}$	FLOPs$_{@640(B)}$
YOLOv5s-seg	640	37.6	31.7	88:16	173.3	1.4	7.6	26.4

Table 4.1: Pre-trained checkpoints

This model consists of a backbone network, a neck and a detection head. The detection head predicts the class and location of objects in the image. The CSPDarknet is used as a backbone network and is responsible for extracting features from the input image; this helps reduce the model's parameters and FLOPS to increase the interference speed and accuracy. The neck consists of a combination of FPN and PANet. FPN transfers the semantic features from the deep layer to the shallow layer to enhance the semantic representation at multiple scales, while PANet transfers the location information from the shallow layer to the deep layer to enhance localization at multiple scales. The deep feature map contains more robust semantic features and weaker localization information, while the shallow feature map contains more robust location information and invalid semantic features.[4].

For the model's training, the loss function indicates the difference between the predicted value and the ground truth. In YOLO v5s, a joint loss function is used to train bounding box regression, classification and confidence. The used loss function is:

$$L = L_{cls} + L_{box} + L_{conf} \tag{4.1}$$

Where L_{cls} indicates the classification error; L_{box} indicates the bounding box regression error; L_{conf} indicates the confidence error.

The classification error L_{cls} is computed by:

$$L_{cls} = \sum_{i=0}^{k^2} I_i E(\hat{p}_i\ (c), p_i(c)) \tag{4.2}$$

when the *i*th lattice of class objects exists, I_i takes 1, otherwise it takes 0. $\hat{p}_i(c)$ and $p_i(c)$ indicate the true probability and predicted probabilty of the *i*th class objects, respectively.

The localization loss L_{box} uses generalized intersection over union loss(GIoU) refers to finding a minimum closed shape *C* such that it can enclose the predicted box A^p with the true box A^g, thus the mathematical expression of L_{box} is:

$$L_{box} = 1 - IoU(A^p, A^g) + \frac{A^c - A^p - A^g + I}{A^c} \tag{4.3}$$

Where A^p indicates the area of the predicted frame, A^g suggests the area of the real frame, A^c indicates the overlapping area of the calculated predicted and real frames, and IoU indicates the intersection ratio of the calculated real and predicted frames.

The mathematical expression of L_{conf} is:

$$L_{conf} = \sum_{i=0}^{K^2} \sum_{j=0}^{M} I_{i,j} E(\hat{C}_i, C_i) - \lambda_{noobj} \sum_{i=0}^{K^2} \sum_{j=0}^{M} (1 - I_{i,j}) E(\hat{C}_i, C_i) \tag{4.4}$$

where K^2 indicates that the input image is divide into K x K grids, and each grid produces M candidate anchors. When the bounding box is a positive sample, $I_{i,j}$ takes 0, and when it is a negative sample, $I_{i,j}$ takes 1. C_i and $\hat{C}_i$ indicate the confidence level of the *i*th predicted bounding and the true bounding box, respectively.

$E(.)$ is a binary cross-entropy loss function, mathematically expressed as:

$$E(\hat{X}_i, X_i) = \hat{X}_i ln(X_i) + (1 - \hat{X}_i) ln(1 - X_i) \tag{4.5}$$

The YoloV5 neural network training method was made and is ideal for object detection generally. When trained with a large open-source labelled dataset, it can look through large datasets of images or videos to identify objects specified to the system or whatever is on a screen. Different versions of YOLO are decided for different resolutions and data sets. We need to detect the apples in an orchard and count the number of apples. This perfectly falls in line with YOLO's speciality. We specifically used YOLOv5s for its efficiency with our small dataset with optimum results. The characteristics were discussed in the table provided above.

5 Experiment and Implementation

As discussed in the research methodology, both the conventional and deep learning approaches worked with Python Programming. A detailed explanation and the general workflow are explained below.

5.1 Conventional Approach

The algorithms used for counting apples in an orchard using the conventional approach are discussed here. The overview of the approach was reading the image first using OpenCV, converting the colour space to HSV, and applying the blob detection algorithm on top of the mask created from the HSV colour space.

5.1.1 Reading image

The libraries were imported in the python file, and the image reading was done using OpenCV

```
import cv2
import numpy as np


image_bgr = cv2.imread("input_image.png")
```

OpenCV reads the image in the BGR format (Blue, Green and Red), so when previewing the image directly without converting to RGB colour space will result in a different colour format than expected.

5.1.2 HSV Color Space

Figure 5.1 represents the HSV (Hue, Saturation and Value) colour space. It is also known as HSB colour space which stands for Hue, Saturation and Brightness.

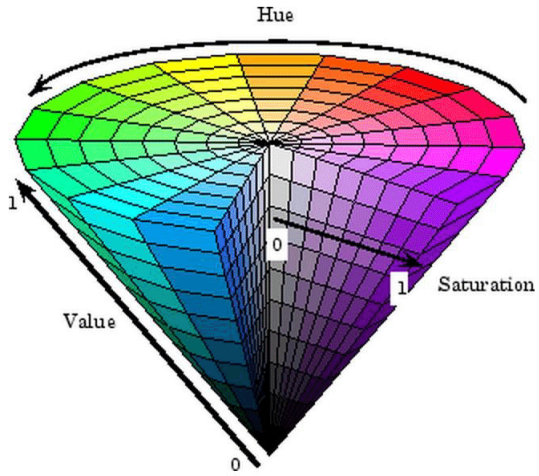

Figure 5.1: HSV Color Space, Image credit: Kemal Erdoğan

It measures how much light is reflected by the pixel. [5]Hue represents the pixel's colour ranging from 0 to 360 degrees. Saturation ranges from 0% (grey) to 100% (saturated) which measures how much white is mixed with the pure colour. Value or brightness ranges from 0% (black) to 100% (white). The high pixel value is brighter, and the low pixel value is darker.

The below code was used to convert the image, which was read in BGR format to HSV format

```
image_hsv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HSV)
```

The separation between H, S and V allows for more accurate colour selection and manipulation since there is significantly less effect of changing light on the HSV colour space. This method is suitable for the apple counting use case as in the orchard, some of the apples in the trees will be directly under sunlight, and others might be in the shady area of leaves. Also, there are different shades of red apples which the apple undergoes to become ripe from raw. This will enable us to create a mask for our application. This mask will be used to isolate the apples, which are red from the background, which is the environment.

To create a mask, we use the following block of code, which accepts two scalar values: upper bounds and lower bounds for H, S and V.

```
low_apple_red = (160.0, 153.0, 153.0)
high_apple_red = (180.0, 255.0, 255.0)
low_apple_raw = (0.0, 150.0, 150.0)
```

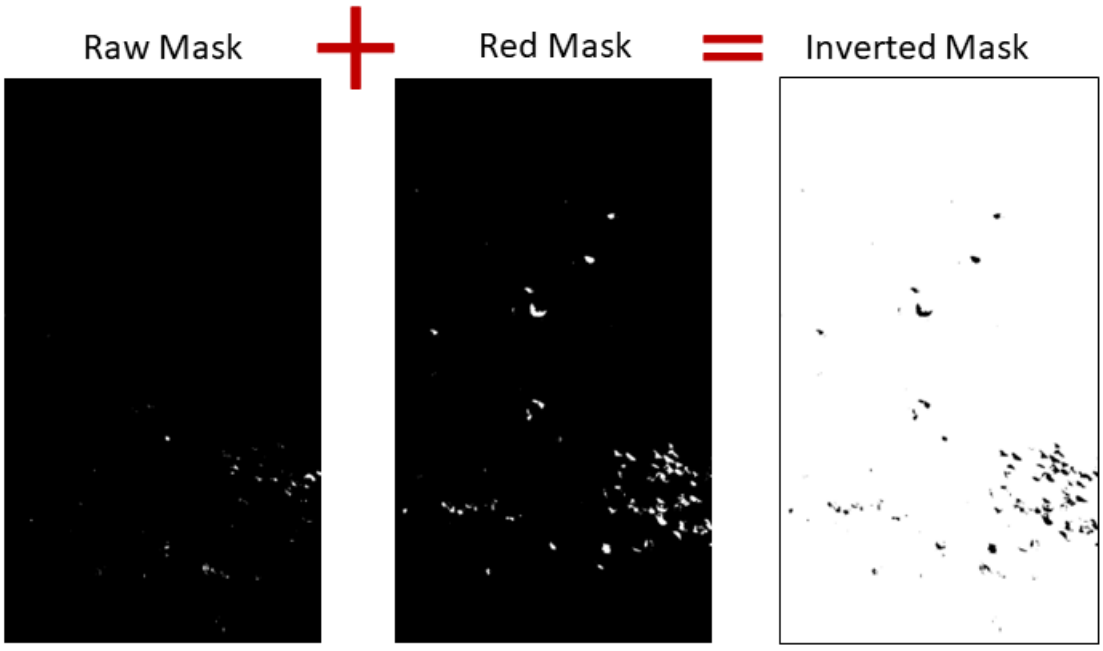

Figure 5.2: Creating of the mask using HSV colour space

```
high_apple_raw = (15.0, 255.0, 255.0)

mask_red = cv2.inRange(image_hsv, low_apple_red, high_apple_red)
mask_raw = cv2.inRange(image_hsv, low_apple_raw, high_apple_raw)

mask = mask_red + mask_raw
mask = cv2.bitwise_not(mask)
```

For the requirement of apple counting, we create two masks: one for raw red apples and the other for ripe red apples. The upper and lower bound of the H, S and V was determined by changing the values in the S and V region after selecting the colour range for red for the H region. Adding both masks will separate the apples of different shades. This will enable us to extract the apples from the background. The mask will isolate the apples removing the rest of the background. As shown in figure 5.2 the mask will have blobs which are the isolated apples from the image. The next step is to count the number of blobs in the mask, which will give us an approximate count of the apples in the image.

5.1.3 Simple Blob Detection Algorithm

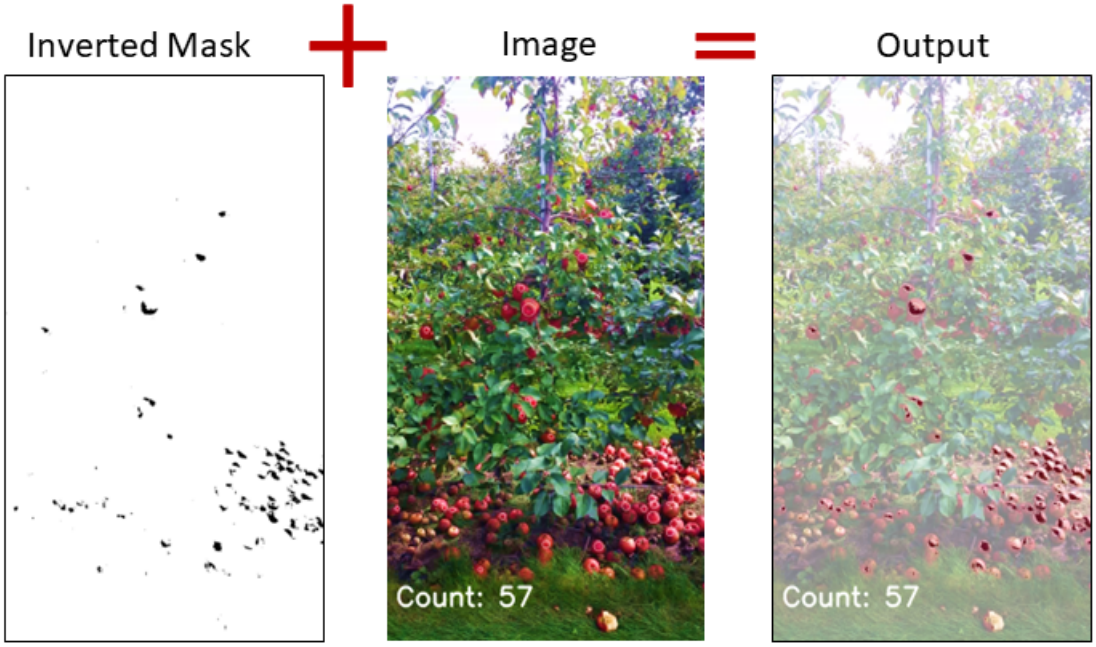

Figure 5.3: Applying the blob detection algorithm to count apples

[6] Blob is a vague shape that does not have any specific shape. Python's SimpleBlobDetector library is a part of the OpenCV computer vision library which detects these shapeless blobs in an image.

This library can detect the key points in an image which are essential. In this case, the region of interest is the location of the circular blobs. These blobs can be of any size or shape and of different colours and shades.

For the use-case of apple counting after applying the mask on the image, the resulting image was just black blobs, as shown in figure 5.2. These blobs were shapeless, and their shade was black only. Each black Blob in the mask represents an apple, as shown in figure 5.3.

The default parameters of SimpleBlobDetector are needed to perform better to detect and count the blobs. So, the library provides some custom parameters that can be changed to improve the blob detection algorithm. Figure 5.4 represents the parameters that can be used. The area puts a threshold on how big or small the Blob should come under the algorithm's detection criteria; thresholds represent the shade of the colour, circularity, inertia, and convexity, focusing on the shape of the Blob.

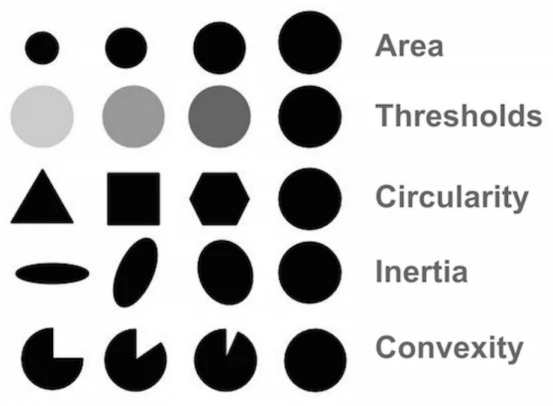

Figure 5.4: Parameters of the SimpleBlobDetector, image credit:Learn OpenCV

The code below shows how the SimpleBlobDetector was implemented:

```
params = cv2.SimpleBlobDetector_Params()

params.filterByArea = True
params.minArea = 20

params.filterByConvexity = True
params.minConvexity = 0.5

detector = cv2.SimpleBlobDetector_create(params)

keypoints = detector.detect(mask)

blank = np.zeros((1, 1))
blobs = cv2.drawKeypoints(image, keypoints, blank, (255, 255, 255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

As per the mask, we got to isolate the apples from the background; it can be seen that the area of the blobs is very small, and the convexity, which is the shape of the blobs, varies. We are not using circularity or inertia as we are not focusing on how circular the blobs are or how many sides the Blob has. So, filtration of the blobs was done using area and convexity, and the parameter values

Figure 5.5: Results of SimpleBlobDetector algorithm

were determined by the hit and trial method to get the optimum performance and reduce the false detection as much as possible.

The parameters were passed to the SimpleBlobDetector algorithm, and then we can get the detected key points of the blobs by calling the method **"detect"** using the object of the algorithm **"detector"**. After storing the locations of the blobs, the key points are drawn as white circles on top of the original image to show where the detections were made, as shown in figure 5.5.

5.2 Deep learning Approach (Using Yolo v5s)

Approach 2 used YOLOv5s to identify the apples in the images and count them. The training images were first loaded on Roboflow for preprocessing, and then labelling was done for around 150 images by hand, then this was augmented further when converting it into a dataset to use for training on Google collab. Our core packages used were PyTorch, roboflow and YOLOv5, IPython.display and OS for operating system interface.

5.2.1 Dataset creation

The acquisition of the current data has been given in the 'Data acquisition and datasets'. Now, the acquired images are loaded onto the *Roboflow* website for labelling and creating a data set.

There are green and red apples in the uploaded images, so we had to make two classes, 'green-apples' and 'red-apples', for the model to distinguish between them. Then, around 150 images were labelled using the Yolo v5s standard. Now, these labelled images are sent for the creation of the dataset.

On the roboflow website, there are multitudes of options that can help us create a dataset with more images than labelled by tilting the present set by a particular angle and in a way, it does not affect the quality or orientation of the objects labelled.

We have downloaded the export *code* to our colab notebook. We can export the generated dataset either by downloading the zip file or by getting a download code for the notebook and letting the program do the downloading for us. It is as follows.

```
!pip install roboflow


from roboflow import Roboflow
rf = Roboflow(api_key="----")
project = rf.workspace("------").project("apple_orchrad")
dataset = project.version(1).download("yolov5")
```

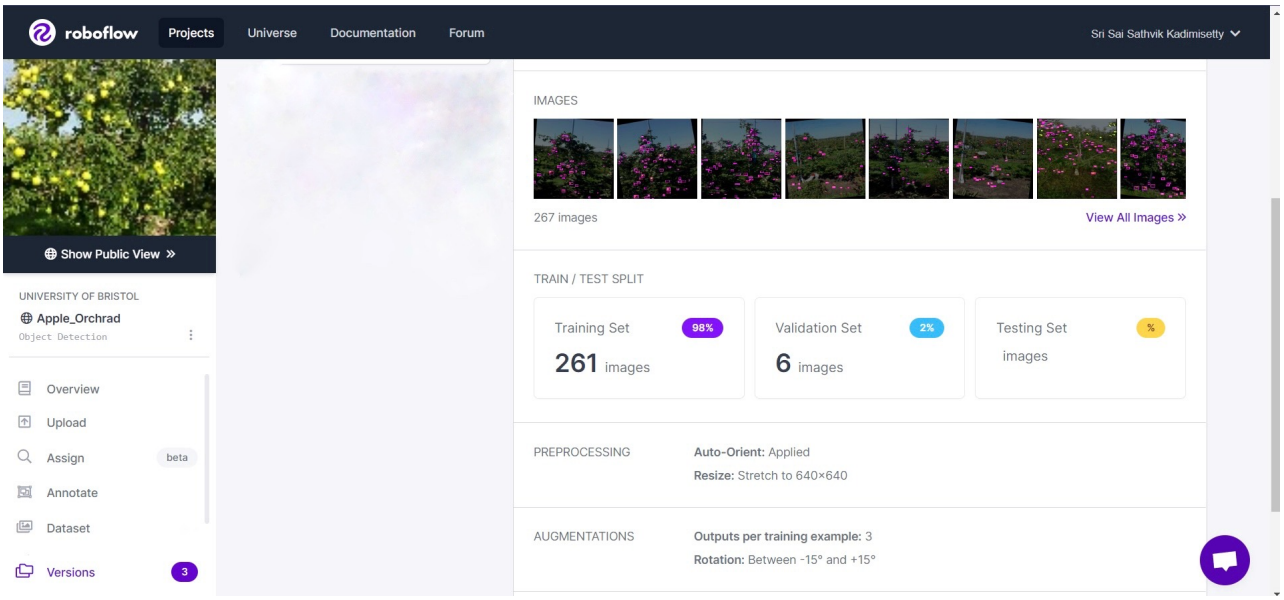

Figure 5.6: Roboflow workspace

5.2.2 Creating the Deep Learning model

To create a YOLOv5s model, we first need to clone the repository into our notebook. The required dependencies must be loaded onto the notebook for the model to work correctly. Then, a variety of packages can be installed based on our preferences. The above description can be shown in the code below.

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images
```

5.2.3 Deep Learning Model YOLO v5s

We first load the dataset from the roboflow workspace to the colab notebook after creating our 'os' environment. We get two folders in the colab workspace 'train' consisting of 261 images, and 'valid' consisting of 6 images.

```
os.environ["DATASET_DIRECTORY"] = "/content/datasets"
```

After this, *train* images are used to train the model. Our dataset resolution was 640 × 640; it was run in batches of 16 and 100 epochs.

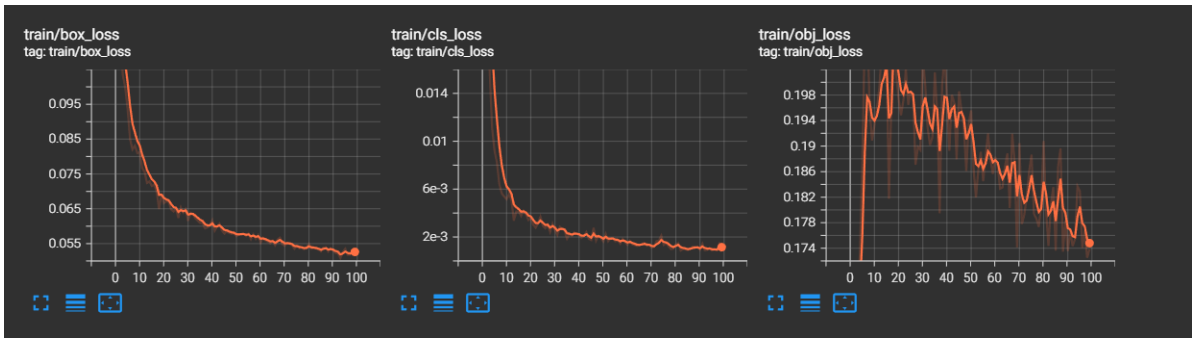

Figure 5.7: Loss Function Graphs

We trained the model on 200 epochs, the Loss Function Graphs were the graph levelled off, and the model was over-fitted. Reducing the epoch to 100 improved the model performance less false predictions, which can be observed from figure5.7.

```
!python train.py --img 640 --batch 16 --epochs 100 --data
/content/datasets/apple-orchard/data.yaml --weights yolov5s.pt --cache
```

The Google Colab notebook, the environment in which we are programming the model, took 19 minutes and 10 seconds to train the model with the images we provided. We used GPU as our Hardware accelerator, which can reduce the amount of time required to train by a significant amount.

As the model is trained for two variants of apple in all sorts of lighting conditions as well as the ripe and unripened conditions, we can now test the model for its *accuracy* and *precession*. We use all the 670 images available for us to test the model. Before running the model, we hypothesised

that using a confidence level of 15 per cent might help us eliminate the false positives generated from using 'green apples' as the objects to detect in an orchard background, the colour being almost the same. The program to detect the number of apples is given below. For this, we have selected the best weight file, which is generated automatically in the YOLO environment is selected. This file is selected as the best weight file consisting of a considerably low loss, and there is no chance of overfitting.

```
!python detect.py --weights
/content/yolov5/runs/train/exp/weights/best.pt --img 640 --conf 0.15
--source /content/test/images
```

After running this, we saw that the model behaved well, eliminating false positives around here. Then the trained model was run on validation and test images to showcase our solution. A specific set of images was selected to showcase our solution's performance diversity for different sets of apples. This was noted down, and corresponding images were displayed and saved to present in this report.

Figure 5.8: Result from running the detection code on a set of images, displaying the number of apples present in a single image.

6 Results and Evaluation

6.1 Evaluation Metrics

Obtained count or output of a deep learning model is first evaluated to obtain the model's Precision, Accuracy and Recall. These metrics help us gauge the quality of our model in a given dataset.

When evaluating a certain outcome of the model, they can be classified into four types; they are as follows.

- **True Positives** are when you predict an observation belongs to a class and it does belong to that class.
- **True Negatives** are when you predict an observation does not belong to a class and it does not belong to that class.
- **False Positives** are when you predict an observation belongs to a class when in reality, it does not.
- **False Negatives** are when you predict an observation does not belong to a class when in fact, it does.

Three main metrics are involved in evaluating a model, which generally helps us compare the results with others and choose the one with better performance. The three metrics are Accuracy, Precision and Recall.

Accuracy is the percentage of correct predictions per test data. It is the division of correct predictions into total predictions.

$$Accuracy = \frac{CorrectPredictions}{AllPredictions} \tag{6.1}$$

Precision is defined as the fraction of the accurate predictions among all the predictions that are said to belong in a particular class.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{6.2}$$

Recall is the fraction of the accurate predictions of a class among the objects that belong to the said class.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{6.3}$$

6.2 Results

6.2.1 Accuracy

The Precision and Recall of the model have been obtained. The Accuracy of the YOLO v5s model we have used above is 84.9 %. These values are tabulated and presented below.

Method	TP	FP	Precision Green Apple	Precision Red Apple	Recall Green Apple	Recall Red Apple	Running Time
YOLO v5s	855	152	88.5	81.8	94.4	82.17	19m 10s

Table 6.1: The Precision and Recall

These values obtained are under the condition of a 0.15 confidence threshold. Here TP indicates True positives, and FP indicates False Positives. The True, False positives of Green and Red apples individually are 408, 53 and 447, 99, respectively.

The Accuracy of the Conventional image processing was obtained by counting the apples in the individual image manually and passing the same image through the conventional function. The count of detected apples was taken, and the Accuracy was found to be 10%.

6.2.2 Precision and Recall

The values of Precision and Recall are calculated individually for the green and red apples. Precision is 88.5 and 81.8; Recall is 94.4 and 82.17, respectively, for Green and Red apples. From the precision and recall values, we can calculate the F1 score. F1 score increases with an increase in precision and recall values and vice versa. The formula for calculating it is given below. The Beta

parameter of the F1 can be varied based on the preference between Recall and Precision. If the Beta is less than 1, Precision is preferred, and Recall is preferred if the Beta is greater than 1.

$$F_\beta = (1+\beta^2)\frac{Precision * Recall}{\beta^2 * Precision + Recall} \tag{6.4}$$

The F1 score is calculated for different values of Beta and is presented below.6.1

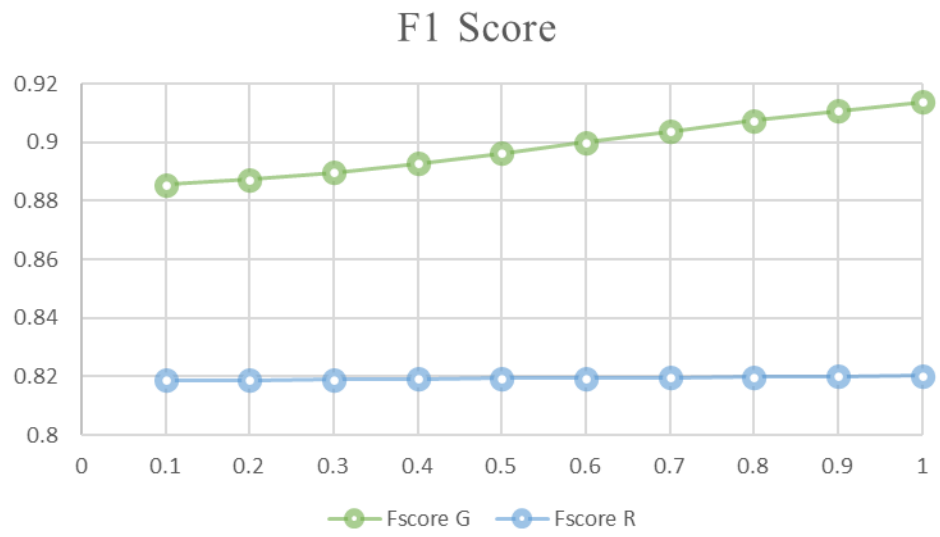

Figure 6.1: F1 Score for Green and Red apples are plotted against Beta

6.3 Evaluation

Conventional image processing and deep learning methods are two techniques used for manipulating digital images. The conventional process relies upon manually designed filters and algorithms to process the image, whereas the deep learning method involves training an artificial neural network on a dataset and making predictions on the image. Figure 6.2a represents the count of apples using the conventional approach, and figure 6.2b gives the count of apples on the same image used for a conventional method using the deep learning method. From the figures, we can evaluate the performance of the conventional v/s deep learning method. In the image, the total count of apples that was done manually was 103 apples, whereas the conventional method detected only seven apples, and the deep learning method counted 109 apples. The conventional method is prone to

environmental lighting, which causes different shades of red apples. Another issue occurs when the apples are in a bunch touching each other, creating a single big red blob, and the third issue is occlusion. In the deep learning approach, we labelled different shades of apples, images with occlusion, and images where apples are in a bunch. This made the neural network learn different situations where apple can be detected as the weight of the neural network is adjusted with features extracted by the model from the image, and we get better-improved results compared to the conventional approach.

(a) Conventional Method

(b) YOLO v5s

Figure 6.2: DIfference between the Apples counted by both the methods used is presented

7 Conclusions and Future works

7.1 Conclusion

7.2 Future work

There are many potential future directions for apple detection methods. We used two methods, out of which the Deep learning method (using YOLO V5) has proven significantly good results and has shown a high accuracy rate and can successfully detect green and red apples separately by overcoming challenges like different lighting conditions and obstacles like leaves.

But we didn't take into consideration about weather and night harvesting scenario. Our methodology can be run on different kinds of datasets produced for different environmental condition and the related data-acquisition methods.

Further studies are suggested with other fruit varieties, of which colour plays a more important role in differentiating them from leaves. Additional fruit attributes such as shape, weight, and colour are also significant information for detecting the market price and are recommended for further investigation.

A Appendix

This is optional. Not every report needs an appendix If you have additional information like code pieces, long tables, etc. that would break the flow of the text in the report, you can put it here.

References

[1] G. Xuan, C. Gao, Y. Shao, *et al.*, Apple detection in natural environment using deep learning algorithms, *IEEE Access* [online], vol. 8 2020, pp. 216 772–216 780, 2020. DOI: 10.1109/ACCESS.2020.3040423.

[2] S. W. Chen, S. S. Shivakumar, S. Dcunha, *et al.*, Counting apples and oranges with deep learning: A data-driven approach, *IEEE Robotics and Automation Letters* [online], vol. 2, no. 2 2017, pp. 781–788, 2017. DOI: 10.1109/LRA.2017.2651944.

[3] L. J. Biffi, E. Mitishita, V. Liesenberg, *et al.*, Atss deep learning-based approach to detect apple fruits, *Remote Sensing* [online], vol. 13, no. 1 2021, 2021, ISSN: 2072-4292. available from: https://www.mdpi.com/2072-4292/13/1/54.

[4] J. Li, Y. Qiao, S. Liu, J. Zhang, Z. Yang, and M. Wang, An improved yolov5-based vegetable disease detection method, *Computers and Electronics in Agriculture* [online], vol. 202 2022, p. 107 345, 2022, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2022.107345. available from: https://www.sciencedirect.com/science/article/pii/S0168169922006536.

[5] R. Python, *Image segmentation using color spaces in opencv + python*. available from: https://realpython.com/python-opencv-color-spaces/.

[6] D. S. Gajbhar, *Detecting and counting apples in real world images using opencv and python*. available from: https://shrishailsgajbhar.github.io/post/OpenCV-Apple-detection-counting.

[2] 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, p. 1222, 2016.