University of the West of England

UFMFRR-15-M

Robotics MSc 2021/22

Machine Vision Final Report

Authors:	*Student IDs:*
Joe Jeffcock	21070345
Connor York	18043219
Harvey Rutland	15002620
Yusuke Miyawaki	21070325
James Reddaway	18013955
Aarsh Dhingra	21070390

Contents

1 Introduction

Machine vision techniques and systems are quickly becoming more common in a variety of working sectors, with agricultural processes being a regular area of research. The use of such systems can provide a variety of benefits, including the monitoring of crops and harvests to improve the efficiency of production and reduce quantities of wasted produce. Additionally, the opportunity for automation made possible from the use of machine vision, can reduce, or remove the need for laborious and repetitive manual tasks such as harvesting, further creating a more efficient process.

This project aims to evaluate both conventional and machine learning approaches of machine vision, specific to the application of detecting and counting apples within an orchard environment. To do this, open-source datasets have been used instead of a manual collection process. Therefore, the assumption that apple datasets are available with images of high enough quality for the evaluation of both methods has been made. Additionally, it has been assumed that dataset examples have been captured in reasonable lighting conditions, for example, not in the dark or adverse weather conditions.

Several challenges have been identified within this project; one such example has involved identifying data sources containing apples within an orchard environment, leading to a dataset with varying backgrounds also being evaluated. Moreover, the varying colour of apples has provided a challenge, as both methods have utilised this data to aid in the detection processes, creating a method specific to apples of one colour in some instances.

2 Related Works

The field of computer vision is fundamental in the development of autonomous systems. In this section we look at relevant literature to inform our development of image processing scripts for apple counting and detection.

Convolutional Neural Networks are a concept of machine learning processes widely used in deep learning for various machine vision problems with high dimensionality such as self-driving cars, facial recognition, and medical diagnosis.

YOLOv3 R-CNN and YOLOv3 are two commonly used deep learning algorithms for detection and segregation of objects. In the paper 'Using YOLOv3 Algorithm with Pre and Post-Processing for Apple Detection in Fruit-Harvesting Robot' (Kuznetsova, 2020). A python based computer vision script is developed for the function of apple detection on a harvesting robot. The standard YOLOv3 algorithm was trained on the COCO dataset for apple detection and tested on a dataset containing 878 (manually acquired) images showing 5142 different (ripe) apple varieties. This gave a test result where 90.9% of fruits were not detected. To improve on this traditional processing techniques such as histogram normalization (used to enhance image detail) and morphological opening (thickening the borders around detection) were implemented prior to running the dataset through the YOLOv3 model. This helped to mitigate issues caused by shadows, glare, apple defects, and overlapping, resulting in increased fruit detection accuracy from 9.1% to 90.8%. The final system was capable of an average apple detection time of 19ms. Although the speed of this application is high, there is clear difference in use context (since we do not seek to control a robot in live operation), use of a fast machine learning model such as YOLOv3 does not provide as much value as high accuracy prediction.

Faster RCNN To further evaluate YOLOv3 we looked directly at the documentation (Redmon, Farhadi, 2018) specifying the accuracy metrics. This presents results from different state of the art machine learning algorithms trained on the COCO dataset, when looking at a metric of mAP at IOU= .5, Faster R-CNN is generally shown to give higher accuracy. The mAP performance can be seen to significantly drop as the IOU threshold increases (a higher threshold requires a higher match of bounding boxes, indicating boxes are not being correctly aligned to the object). It is also noted that YOLOv3 struggles with smaller objects, this may lead to issues when using this model to count apples in an orchard environment. Due to prediction accuracy being more beneficial to this project compared to processing speed, the Faster RCNN alternative appears to be better suited to the task of counting apples in an orchard environment.

Region-based Counting (Chen, 2017), proposes a Machine Learning architecture to count apples and oranges in images captured via mobile drones. The architecture utilises a Fully Convolutional Layer (FCN) to generate segmentations of apples and oranges in images, where extracted blobs can represent one or many overlapping fruits. Each blob is further passed through an FCN to predict how many fruits exist within them. The method achieves state-of-the art results on the fruit counting task, however it was deemed inappropriate for this work as the dataset is not provided by the authors, while preparing a custom dataset would prove time-consuming given the project scope.

Simple Online Realtime Tracking (SORT) Multiple Object Tracking (MOT) attributes identities to objects detected in images over multiple frames. Simple Online and Realtime Tracking (SORT) is an MOT algorithm that models size and movement of each detected object's bounding box using Kalman filters. At each frame, SORT's Kalman filter models predict updated positions and shapes for tracked apples, while a base object detection method localises objects present in the image. Detections are matched to tracked objects using a maximum matching algorithm on the bounding boxes, where unmatched detections are added to MOT and unmatched tracked apples are dropped from MOT.

By assigning identities to each object, the SORT algorithm inherently produce a tally of objects detected. We feel the algorithm can increase versatility of the proposed solution by enabling apple counting over moving images captured by a mobile robot, without repeat inclusion of previously seen apples.

3 Data Acquisition and Datasets

3.1 Data Acquisition

Infra-red Sensors Research in the field of machine vision highlights several processes and sensors suitable for the application of apple counting. One such method is the use of infra-red (IR) time-of-flight sensors, which operates as an active sensor, emitting IR light into the surrounding environment and calculating distances using the phase shift between illumination and reflection (Li, 2022). Li (2022) describes that this type of sensor provides a low-cost solution, that is, however, often limited to structured environments such as industrial scenarios with applications including tasks like object counting from a conveyor belt.

LiDAR Cameras LiDAR cameras provide an alternative, allowing for real-time detection with the formation of high-definition 3D models. Similar to the process of IR sensors, modulated light is emitted, and the phase shift is used to calculate distances, with the added difference that this method is capable of synchronously measuring multiple light beams to achieve angular data (Wang, 2021). However, Wang (2021) highlights that required hardware is costly and the methods required are often computationally expensive. Moreover, in the scope of this project, it could be considered that 3D data capture is unnecessary.

Cameras Cameras are presented as one of the most popular methods, with images collected by exposing an array of photosensitive cells to light from the surrounding environment. Wang (2021) describes the use of cameras as a cheaper and less computationally expensive alternative to LiDAR. Additionally, a wide range of open-source data sets containing numerous labelled images (RGB, HSV etc), are regularly available online. As datasets are easily obtainable, this project has evaluated machine vision techniques using datasets such as MinneApple (Häni, Roy and Isler, 2019) and MSCOCO (Cocodataset.org, 2020).

3.2 Datasets

Fuji 3D The Fuji 3D dataset uses a series of high-quality images to build 3D models of 11 different Fuji apple trees, containing 1455 separate instances of apples (Gené-Mola et al., 2020). Although 3D data has not been used in this project, this dataset has been included in the discussion as it provides a strong basis for future work. 2D images are also made available through the dataset, however the sample size of 11 trees was deemed too limited for the 2D object detection task.

MinneApple The MinneApple (Häni, Roy and Isler, 2019) dataset contains 1000 images with over 41,000 separately labelled instances of apples (between 1 and 120 per image) (Häni, Roy and Isler, 2019). Additionally, the images of this dataset have been captured in an orchard environment (as shown in Figure 1), making this dataset highly relevant to the application of this project. However, mainly due to the number of apples per image, the quality of each instance is relatively low (Figure 1), providing less information for machine learning approaches. Within this project, this dataset has been used for the training of a machine learning model, as well as the testing for both conventional and machine learning methods.

MSCOCO When using MSCOCO 2017 (Cocodataset.org, 2020), the dataset has been filtered to only include instances of apples, providing an approximate 1600 images of varying quality. However, unlike the MinneApple dataset, instances of apples are contained in variable environments, with most images not captured in orchards (example image shown in Figure 2), making this a less suitable dataset. From this dataset, a machine learning model has been trained for evaluation.

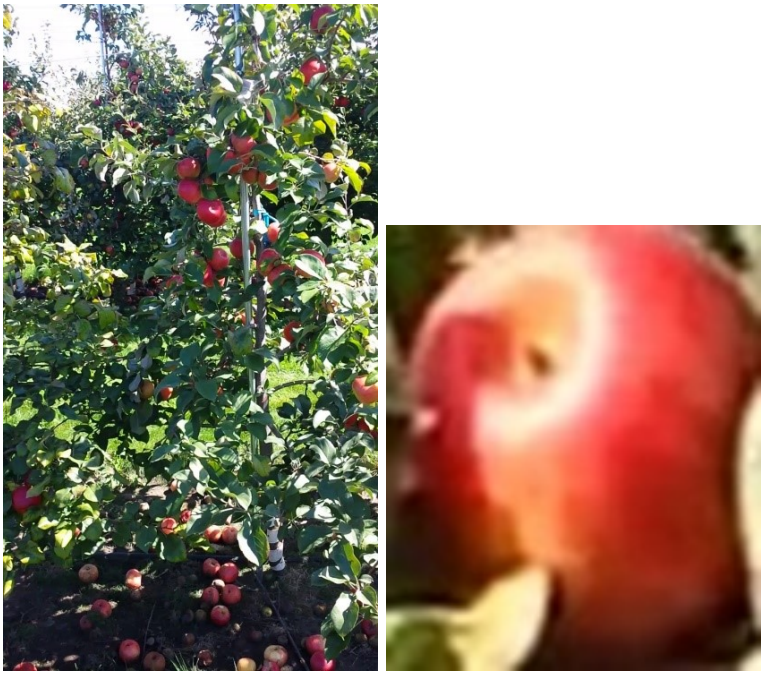


Figure 1: MinneApple Sample Images (Häni, Roy and Isler, 2019)

Figure 2: MSCOCO Sample Image (Cocodataset.org, 2020)

4 Methodology

In our methodology, we pose the problem of apple counting as the tally of apples detected in an image. Following this definition of the problem, we propose one conventional machine vision approach and one machine learning method to detect apples in images.

4.1 Conventional Approach

In this paper, the method using opening and closing is adopted as a conventional approach. The flowchart of this method is shown in Figure 3. As illustrated, this method has three pre-processing operations, namely conversion to HSV colour space, extracting colours from the original image, and conversion to greyscale. Then, the opening and closing is applied to remove noise from images. Finally, using the processed image, the number of apples in the original image is counted and displayed. The details of each process will be described below.

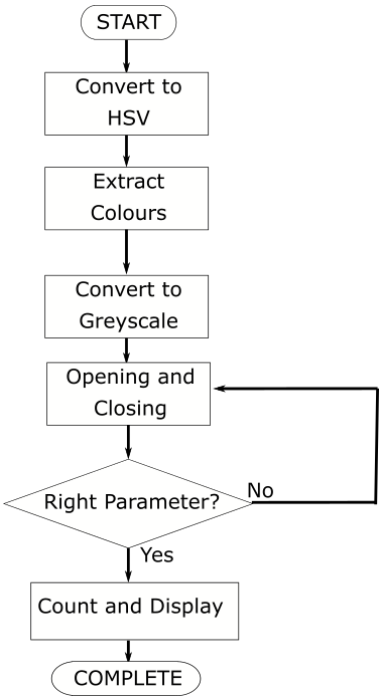


Figure 3: Flowchart of the Conventional Method

In pre-processing, objects need to be highlighted. The task is to detect apples from an orchard, therefore applying masks to extract colours of apples from images can be useful. By setting minimum and maximum colours of the mask and applying it to the original image, only the colours of apples are highlighted. By using HSV images, saturation and brightness can be controlled, and the algorithm can be more robust than using RGB images. The parameter of the minimum and maximum colour of the mask needs to be adjusted depending on the colour of apples and background. Then, the image mask is converted to greyscale to apply the threshold.

Background removal in RBG images can be achieved with binarization and Otsu thresholding, which takes the variance between clusters as the criterion to select the optimal threshold (Zhu, Wang, Yang and Dai, 2009). However, in this report, this was not required, as we found that the manual thresholding of saturation and brightness in HSV images was sufficient in removing backgrounds to produce an optimal pre-processed image (Figure 8).

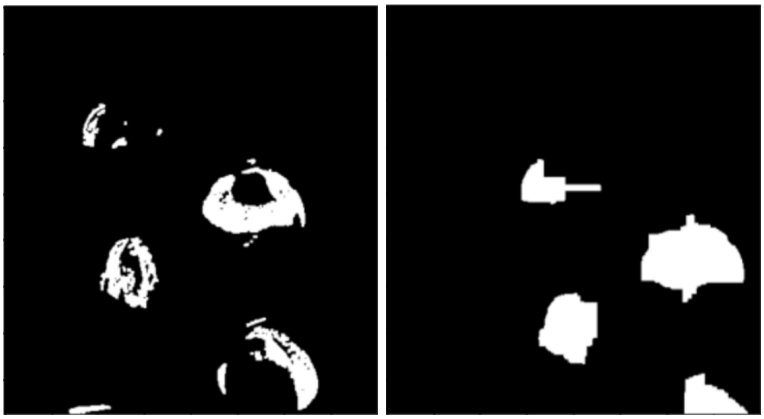
The next process is the opening and closing, which serves two purposes. The first purpose is removing the white noises other than objects, for example, the pre-processed image shown in Figure 4a has white

noises around objects. However, there is potential that this noise could be counted as objects. To remove this noise, an opening operation is applied, as shown in Figure 4.

(a) Binary image before opening (b) Binary image after opening

Figure 4: Example of Opening

The second purpose of this process is to fill black holes in objects. For example, the image shown in Figure 5a displays these types of black holes in the object, however, some of these holes can create separation and cause multiple counts for a single apple. So, a closing process is needed when a pre-processed image has black holes in objects. Figure 5 shows the image closing process, with the black holes in Figure 5b having been removed through this process.



(a) Binary image before closing (b) Binary image after closing

Figure 5: Example of Closing

Opening and closing are the combination of dilation and erosion, with the opening process applying erosion followed by dilation and being effective in removing noise. On the other hand, closing is the operation of dilation followed by erosion, and this is effective in filling small black holes in objects.

Erosion is a process to remove pixels that are adjacent to the background, as shown in Figure 6. Erosion can be operated by itself, to do this, a kernel needs to be set and when black pixels exist in the kernel, the focused pixel becomes black.

Dilation is the process to add new pixels around the boundary of a region, as shown in Figure 7. Dilation can be operated by itself, and like erosion, a kernel needs to be set, with the difference that when white pixels exist in the kernel, the focused pixel becomes white.

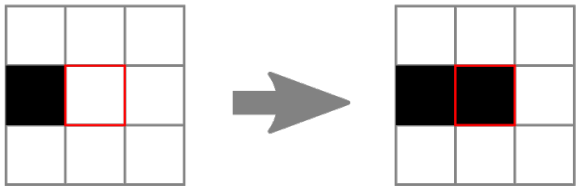


Figure 6: Erosion

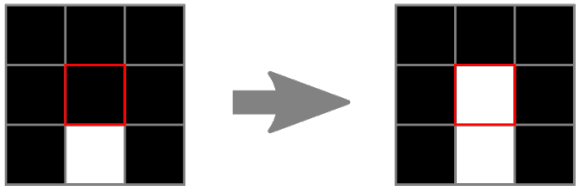


Figure 7: Dilation

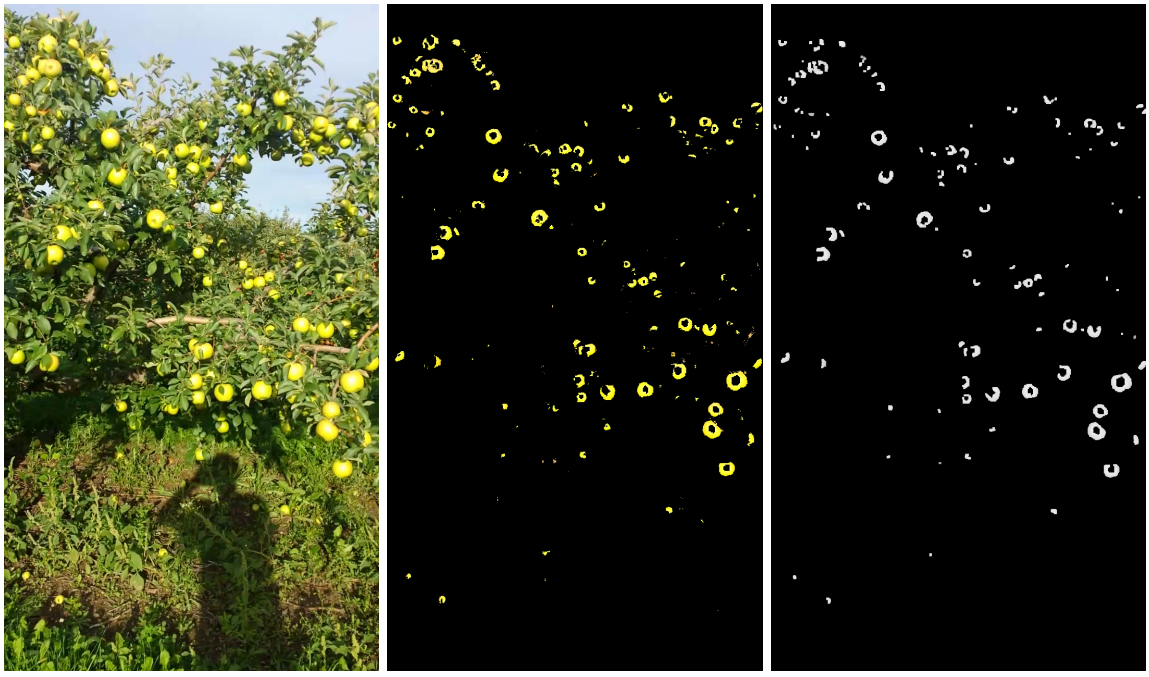
The number of the iterations and kernels of opening and closing depend on the original images. After opening and closing is applied, the image should contain the same number of white objects as instances of apples in the original image. Therefore, by counting the number of objects, the number of apples in the images can be obtained. To optimise this method, optimal threshold values, kernel size, and number of iterations of morphological operations needs to be found. This is difficult however as they are highly dependent on the original image and therefore may not transfer well between images. In this task, it is expected that objects do not contain black holes which separate the objects within them, as in most cases as shown in 8. As such, our approach does not use closing and relies only on opening to remove noise.

4.2 Machine Learning Approach

This paper looks at training 2 machine learning models on the Minnesapple and COCO datasets both using Faster-RCNN.

In Faster R-CNN, an image is passed to a convolutional neural network (CNN), returning a feature map of the image. A proposal network is then applied to the feature maps, returning the highlighted regions along with an objectness score. A novel region of interest (ROI) pooling is then applied to resize the selections before passing them to a softmax layer and linear regression layer for classification and prediction of bounding boxes, respectively.

The initial step of faster RCNN uses an initial set of convolutional layers to learn the feature map with the RPN (depicted in Figure 8). An example of this is ResNET-50 (a CNN consiting of 50 layers) this is a common backbone in many computer vision tasks. Use of a Region Proposal Network (RPN) in conjunction with ResNet has shown improvement on previous selective search methods used in Fast-RCNN. ResNet50 is provided in certain computer vision programming libraries (such as PyTorch).



(a) Original image (b) Image after HSV thresholding (c) Binary image after opening

Figure 8: Example of Task

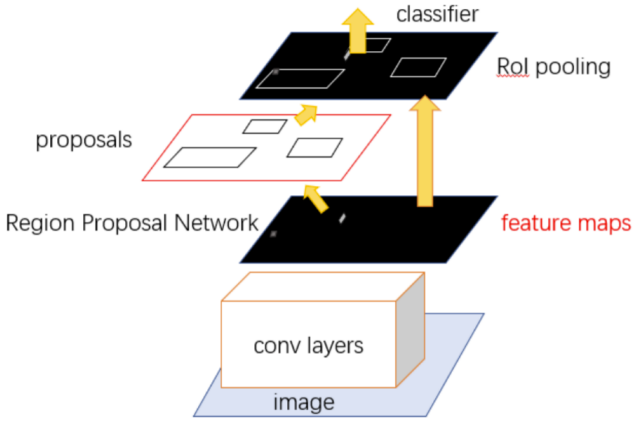


Figure 9: Faster R-CNN Model - a unified object detection network (Ren et al., 2015)

A forward pass of Faster R-CNN during training involves the calculation of 4 loss functions to measure error between predictions and ground truth, comprising smooth L1 losses for bounding box regression on each of region proposals and detections, a binary cross-entropy loss on "objectness", and cross-entropy loss on object classification. Minimising smooth L1 losses result in bounding boxes that are of lower distance from ground truth, while cross-entropy losses express classification tasks as probabilities, where labels corresponding to ground truth exhibit higher probabilities on minimisation.

Backpropagation of the loss allows us to observe the gradient at each parameter of the network in relation to the loss, indicating the direction of steepest ascent of the loss at each parameter and the magnitude of its contribution. This is achieved by recursively applying the chain rule from the final layer backwards, where the derivatives of simpler, composite expressions can be compounded to obtain gradients at each term of a complex function.

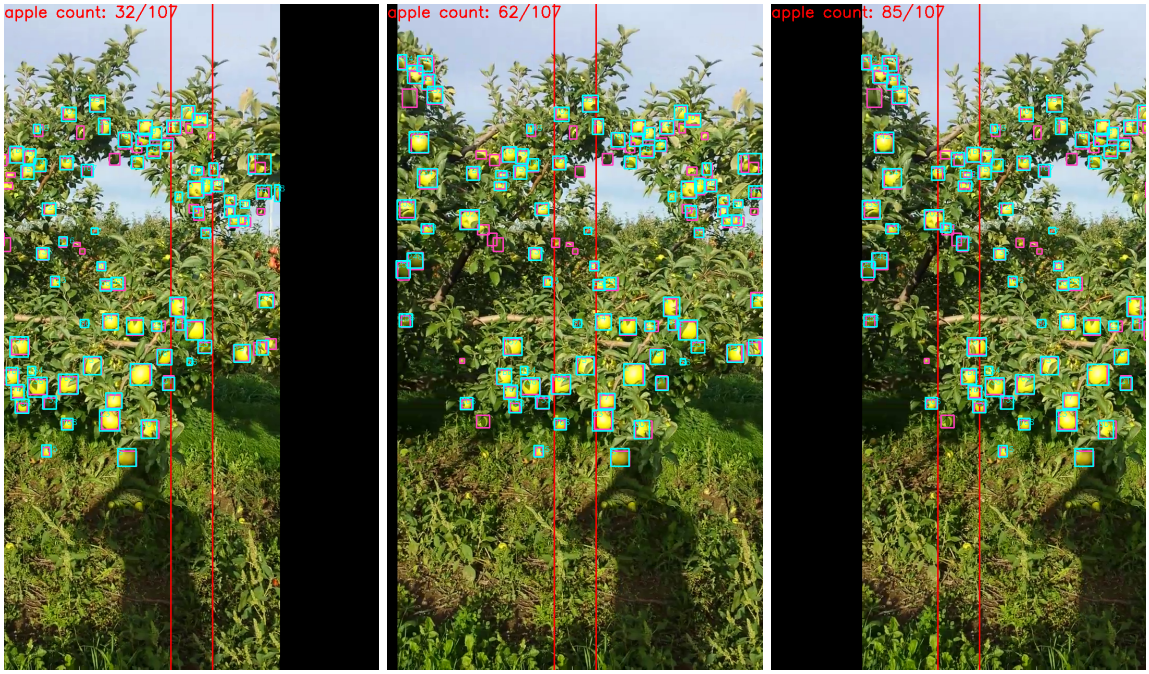
Faster R-CNN is trained iteratively by Stochastic Gradient Descent, where the average loss is computed for a batch of training data and backpropagation is used to obtain the gradient at each parameter of the network. By iteratively subtracting the gradient, multiplied by a scalar learning rate, the weights of the network take steps in the direction of steepest descent of the error, enabling Faster R-CNN to approximate a function of the apple detection task.

4.3 Extension: Multiple Object Tracking

The sections above outline conventional and machine learning approaches to object detection, useful for tallying apples in a single image. However, we expect motion to be a vital component of a robotic application for apple counting, where a mobile robot captures apples across many frames. Image capture at high framerates will result in apples being counted repeatedly when captured over multiple frames, while low framerates may prevent apples from being captured in-frame as the robot moves past them.

We propose to overcome this issue using MOT, where detected apples are tracked across frames in realtime to prevent repeated counts.

To achieve counting, the number of unique IDs generated from the SORT tracking algorithm can be taken as the tally of apples. This approach, however, is prone to over-counting as SORT is highly susceptible to fragmentation, where tracking is temporarily interrupted, and a new ID is generated for the same apple. We mitigate this by parameterising a region in the image where counting takes place (Figure 10), effectively limiting the number of frames over which fragmentation can occur during motion.



(a) (b) (c)

Figure 10: Left to Right: Count of Apples using MOT in a Moving Image, Border Size = 80

5 Experiment and Implementation

5.1 Experiment

We prepare evaluation scripts on the MinneApple dataset to test the proposed methods. 100 images not used for training or validation are taken from the MinneApple dataset as our test set, and for each image and ground truth (GT) we accept detections as inputs of either a list of points for conventional methods, or a list of bounding boxes for Machine Learning methods. Metrics are computed based on GT and detections for each image to yield results for the experiment, further explained below and in Section 6.1.

In evaluating a given image, we compute the minimum matching of conventional method detections to GT by distance between detected points and GT bounding box centres, matching detections to the closest bounding box. We compute the maximum matching of machine learning detections to GT by the intersection over union (IOU) (Figure 11) between detected bounding boxes and GT bounding boxes, matching detections to GTs with which they have the most overlap. These matchings represent True Positive (TP) detections, where matches are only kept if there is a non-zero overlap between detections and GT. Detections that are not matched comprise False Positives (FP), while GT instances not matched comprise False Negatives (FN).

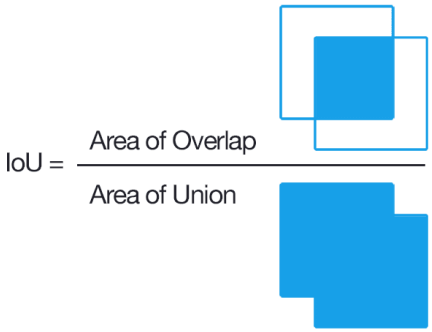


Figure 11: Equation of Intersection over Union (IOU) (Rosebrock, 2016)

Figure 12 shows the implementation of matching conventional method detections to GT using `numpy` and `scipy.optimize.linear_sum_assignment()`. The matching for machine learning follows a similar pattern, however using `torchvision.ops.box_iou()` in place of the distance matrix.

Using TP, FP, and FN, we compute Precision, Recall and task-specific metrics for apple counting using conventional and machine learning methods. A discussion of the metrics can be found in Section 6.1.

Extending the conventional and machine learning methods, we prepare an evaluation script for MOT-based counting. To our knowledge, no dataset exists for MOT in apple orchards, and as such we adapt the evaluation process above to test MOT performance. We use the same test set of 100 images and simulate motion by translating images across the frame in 5px steps, updating the SORT algorithm with new detections at every step. A region of frame-height and a parameterised width is defined at the centre of the frame, where the IDs of apples that fall in this region during motion are added to a set (Figure 13). The size of the set at the end of an image's motion is taken as TP.

We select two variations of each method to evaluate on the test set. For the conventional approach, we perform a parameter search over 100 configurations of closing kernel size and kernel iterations ranging from (1,10) each. The conventional configurations are first evaluated on the validation set, where the highest performers on each of total and average metrics are selected for evaluation on the test set. Faster R-CNN models trained on the MinneApple and COCO dataset are chosen for evaluation of the Machine Learning approach on the test set. Similarly, MOT is evaluated on the test set using detections provided by each of the Faster R-CNN models.

```python
def dist_points_from_box_centres(points, boxes):
    boxes_np = np.array(boxes)
    centres = np.zeros((len(boxes), 2), dtype=np.float32)
    centres[:,0] = (boxes_np[:,0] + boxes_np[:,2])/2.0
    centres[:,1] = (boxes_np[:,1] + boxes_np[:,3])/2.0

    centres = np.expand_dims(centres, 0)
    points = np.expand_dims(points, 1)

    dist_matrix = np.linalg.norm(points - centres, axis=2) # l2 distance
    return dist_matrix


def match_points_to_box_centres(points, boxes):
    # calculate distance (cost) matrix
    in_matrix = test_points_in_boxes(points, boxes)
    dist_matrix = dist_points_from_box_centres(points, boxes)

    # set high cost for points outside of each box
    float_max = np.finfo(dist_matrix.dtype).max
    dist_matrix[np.invert(in_matrix)] = float_max

    matched_indices = linear_sum_assignment(dist_matrix)

    matches = []
    for k in range(matched_indices[0].size):
        i = matched_indices[0][k]
        j = matched_indices[1][k]
        if(in_matrix[i,j]):
            matches.append((points[i], boxes[j]))

    return matches
```

Figure 12: Code: Matching Conventional Approach Detections to Ground Truth

```python
            # update sort
            if len(dets):
                tracks = sort.update(np.array(dets))
            else:
                tracks = sort.update(np.empty((0, 5)))

            # add identities within user-defined borders
            for track in tracks:
                x1, y1, x2, y2 = [int(v) for v in track[:4]]
                cx = (x1 + x2)/2
                identity = track[4]
                if cx > border[0] and cx < border[1]:
                    identities.add(identity)
```

Figure 13: Code: Storing Detections Tracked by SORT

5.2 Implementation

5.2.1 Conventional Approach

The code in 14 is used to extract specific HSV of apples from original images. To extract HSV, applying a mask which has minimum and maximum parameters is needed. As argued in the section 4.1, the parameter depends on the colour of apples and lighting of an original image so, adjustment is needed in this part.

```
12  #Extract colours From Image
13  img_mask = np.zeros(img.shape[:2], dtype=np.uint8)
14  for lower, upper in thresholds:
15      bgrLower = np.array(lower) # minimum color(HSV)
16      bgrUpper = np.array(upper) # maximum color(HSV)
17      curr_mask = cv2.inRange(img_hsv, bgrLower, bgrUpper) # make mask
18      img_mask = cv2.bitwise_or(img_mask, curr_mask) # combine mask
19  result = cv2.bitwise_and(img, img, mask=img_mask) # apply mask
```

Figure 14: Code: Extracting HSV Ranges from an Image

The code in 15 is used to apply opening to a image. In this part, the kernel size and the number of iteration need to be decided. The number of iteration depends on the level of noises. If the iteration is too few, the noises may remain on the image but if the iteration is too many, the target objects may be removed from the image.

```
24  # opening
25  kernel = np.ones((kernel_size, kernel_size),np.uint8)
26  opening = cv2.morphologyEx(gimg, cv2.MORPH_OPEN, kernel, iterations=iterations)
```

Figure 15: Code: Apply Opening to a Mask

After the opening process, zero values are considered as the background and non-zero values are considered as objects. Therefore, using the code in 16, the features in 2D array image are counted as the number of apples.

```
28  labels, nlabels = ndimage.label(opening)  # Label features in an array. Any
29  centroid = ndimage.center_of_mass(opening, labels, np.arange(nlabels) + 1 )
30
31  predictions = []
32  for point in centroid:
33      y, x = point
34      predictions.append([int(x), int(y)])
35  return(predictions)
```

Figure 16: Code: Labelling and Extracting Detections

5.2.2 Machine Learning Approach

Both the Minnesapples and COCO (fiftyone used to select only images with the tag 'apple') datasets were downloaded locally. The code in Figure 17 is used to load and select the datasets for training, in addition errors are raised if the dataset is not one that has been specified. The class number is set to 2 as only background and apple classifications require highlighting.

The Pytorch library was used to train the machine learning model. Pytorch is an open-source deep-learning framework, with simple implementation for computer vision applications (Pytorch.org, 2022). Figure 18 depicts the torch.utils.data.DataLoader class, the dataset variable refers to the dataset we desire to train. Other parameters such as shuffle and batch size alter the order in which the images are presented for training (ensuring variation over loaded samples) and dictates the number of images used to train the model at a given time. The parameter num workers enables multiprocessing and dictates the number of batches which are trained simultaneously.

```
# our dataset has two classes only - background and APPLE
num_classes = 2

# use our dataset and defined transformations
if args.dataset == "MinneApple":
    dataset = MinneAppleDataset(args.train_data_path, get_transform(train=True))
    dataset_test = MinneAppleDataset(args.val_data_path, get_transform(train=False))
elif args.dataset == "COCO":
    # first build the 'fiftyone' datasets
    dataset_51 = get_fiftyone_dataset(args.train_data_path)
    dataset_test_51 = get_fiftyone_dataset(args.val_data_path)

    dataset = CocoDataset(dataset_51, get_transform(train=True))
    dataset_test = CocoDataset(dataset_test_51, get_transform(train=False))
else:
    raise ValueError("Dataset: {} is not supported".format(args.dataset))

```

Figure 17: Code: Selection of dataset path

```
    # define training and validation data loaders
    data_loader = torch.utils.data.DataLoader(
        dataset, batch_size=1, shuffle=True, num_workers=4,
        collate_fn=utils.collate_fn)

    data_loader_test = torch.utils.data.DataLoader(
        dataset_test, batch_size=1, shuffle=False, num_workers=4,
        collate_fn=utils.collate_fn)
```

Figure 18: Code: Selection of dataset path

Transformations are performed on images to manipulate the data and provide more variation when training the machine learning model. Figure 19 depicts a section of the script used for transformations in this project. These functions come from the torchvision package are commonly used in computer vision applications (Pytorch.org, 2022). Of the operations listed the first "transforms.append(T.ToTensor())" converts the image to tensors (This is the data format used yo train the model). The second "transforms.append(T.RandomHorizontalFlip(0.5))" is used to flip the image data horizontally, this adds variation in the training dataset, the value "0.5", refers to implementation of this transformation on 50% of the dataset allocated for training ('Shri R', 2021).

```
def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

Figure 19: Code: Labelling and Extracting Detections

Prior to training the machine learning model, several hyper parameters are set which control the learning process configuration. An example of this is selection of the optimisation algorithm, this modifies certain weight parameters of the neural network, to minimize prediction error between batches. In this example the torch.optim package is used to implement the SGP algorithm (for gradient decent) and lr_schedular (declaration of loading rate) (Ayush Gupta, (2021)).

The epoch number refers to the number of times the entire training dataset is passed through the learning algorithm. Figure 20 shows 10 epochs, to be used in the training model (meaning 10 iterations of the specified batch number). Between epoch iterations the loss value decreases (loss being an indicating of the model's prediction with a perfect value of 0). The epoch value was selected by initially training the model and

observing the loss value, because the loss value could not be seen to greatly decrease after 10 epochs the epoch value was set to 10. Specifying an epoch value to be higher than what is necessary can lead to overfitting, where the model is only able to perform on the training dataset, and does not give the desired outcome on unseen datasets.

```
54    # create model and load it onto device
55    model = get_model_faster_rcnn(num_classes)
56    model.to(device)
57
58    # construct an optimizer
59    params = [p for p in model.parameters() if p.requires_grad]
60    optimizer = torch.optim.SGD(params, lr=0.005,
61                                momentum=0.9, weight_decay=0.0005)
62    # and a learning rate scheduler
63    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
64                                                   step_size=3,
65                                                   gamma=0.1)
66
67    # let's train it for 10 epochs
68    num_epochs = 10
69
70    for epoch in range(num_epochs):
71        # train for one epoch, printing every 10 iterations
72        train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
73        # update the learning rate
74        lr_scheduler.step()
75        # evaluate on the test dataset
76        evaluate(model, data_loader_test, device=device)
77
78    torch.save(model.state_dict(), './weights.pth')
```

Figure 20: Code: Labelling and Extracting Detections

After training the models had been trained on the COCO and Minnesapple datasets. The quality of the output was checked through manual selection of a random image from the test dataset. Visual inspection was used to initial inform whether the hyper parameters used to tune the model needed adjusting. After the performance of both models had been validated they were loaded into the evaluation script to allow for performance to be compared between conventional and machine learning methods.

5.2.3 Multiple Object Tracking

The MOT extension of our project is achieved using the SORT implementation provided by Alex Bewley. Bounding boxes detected using machine learning are converted to numpy arrays and passed to an instance of SORT by the object's `update()` function (Figure 21). `update()` returns an array of bounding boxes and corresponding IDs representing the apples currently tracked by the algorithm.

```
58        # NN predictions
59        predictions = model([img_affine.to(device)])
60        pred_boxes = predictions[0]['boxes'].cpu().detach().numpy()
61        pred_labels = predictions[0]['labels'].cpu().detach().numpy()
62        pred_scores = predictions[0]['scores'].cpu().detach().numpy()
63
64        # non-maximum suppression
65        indices = torchvision.ops.batched_nms(predictions[0]['boxes'], predictions[0]['scores'], predictions[0]['labels'], 0.3)
66
67        # filter by score
68        dets = []
69        for index in indices:
70            x1, y1, x2, y2 = [int(v) for v in pred_boxes[index]]
71            score = pred_scores[index]
72            area = (x2 - x1) * (y2 - y1)
73            if score > 0.1:
74                dets.append([x1,y1,x2,y2,score])
75
76        # update sort
77        if len(dets):
78            tracks = sort.update(np.array(dets))
79        else:
80            tracks = sort.update(np.empty((0, 5)))
```

Figure 21: Code: Passing Detections to SORT

6 Results and Evaluation

6.1 Metrics and Results

We compute the following metrics for analysis, using the tallies of predicted and ground truth apples, as well as TP, FP and FN described in Section 5.1.

- **Precision = TP/(TP + FP):** A common metric used in Machine Learning that indicates the proportion of detections that are relevant to ground truth.
- **Recall = TP/(TP + FN):** A common metric used in Machine Learning that indicates the proportion of ground truth apples detected by the model.
- **Relative Error =** $\frac{count(predictions)-count(ground\_truth)}{count(ground\_truth)}$**:** Describes the error of the tally of predicted apples relative to the tally of ground truth.
- **Absolute Relative Error =** $abs(relative\_error)$**:** Describes the magnitude of the relative error for use in averaging of metrics and computing the counting accuracy.
- **Counting Accuracy =** $1.0 - abs(relative\_error)$**:** Describes the relative closeness of the tally of detected apples to the tally of ground truth - a direct measure of success for the apple counting task.

The metrics are computed for each method on the test set and tabulated in Tables 1-4. Total metrics refer to the metrics computed on the total TP,FP and FN obtained across the entire test set, while average metrics refer to the average of metrics computed on TP,FP and FN obtained per image.

Qualitative results are presented in Figures 22, 23 and 24, where ground truth bounding boxes appear in pink and predictions appear in light blue.

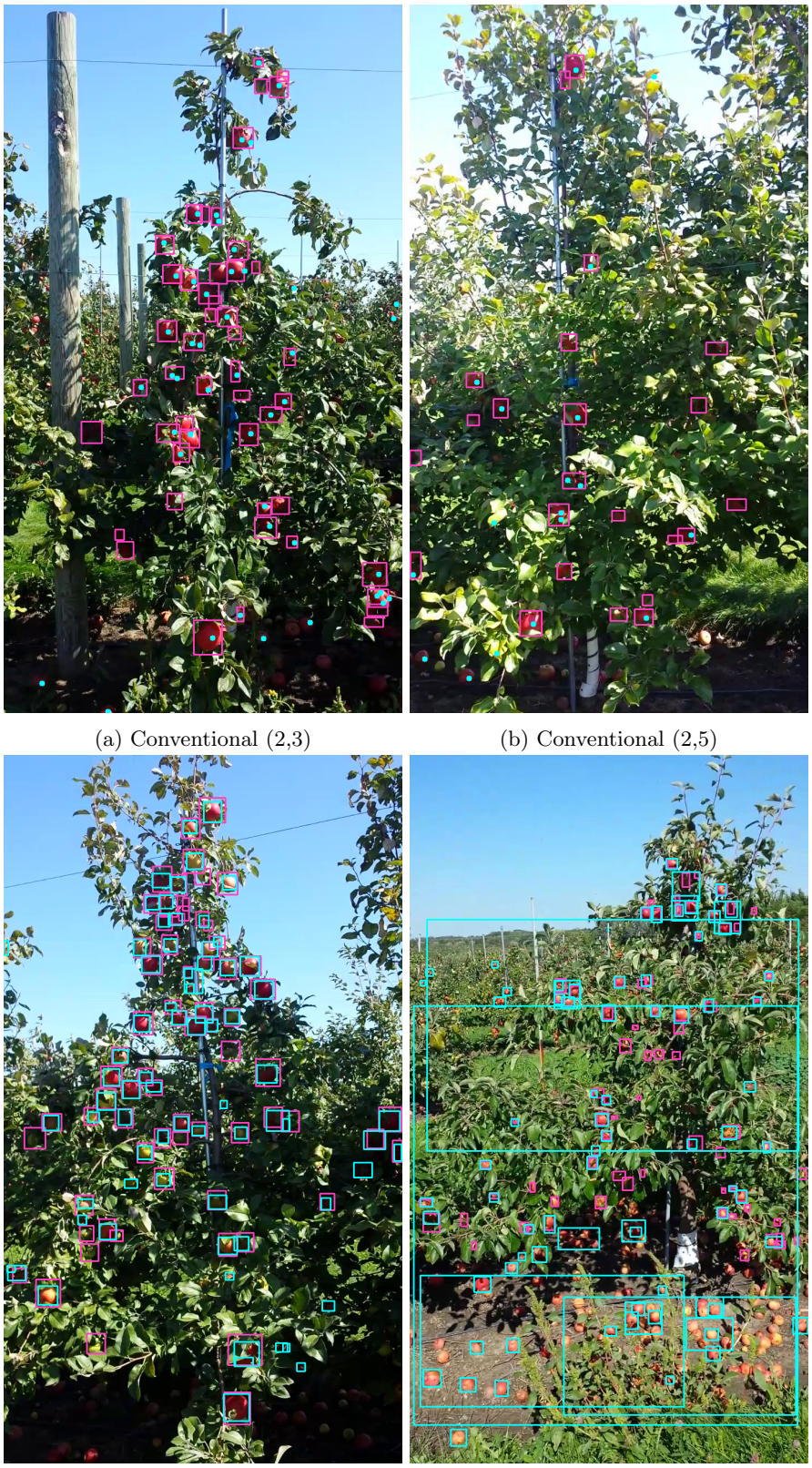
Conventional (k, i) refers to the conventional approach with kernel size k and i iterations, while Faster R-CNN (X) refers to the machine learning approach trained on dataset X.

Method	TP	FP	FN	Precision	Recall	Relative Err.	Counting Acc.
Conventional (2,3)	1955	1521	1893	0.562	0.508	**-0.097**	**0.903**
Conventional (2,5)	1361	**576**	2820	0.703	0.326	-0.537	0.463
Faster R-CNN (MinneApple)	**3772**	1153	**409**	**0.766**	**0.902**	0.178	0.822
Faster R-CNN (COCO)	2908	2520	1273	0.536	0.696	0.298	0.702

Table 1: Total Metrics for each Method across the Test Set

Method	Precision	Recall	Relative Err. (abs)	Counting Acc.
Conventional (2,3)	0.493	0.481	0.701	0.300
Conventional (2,5)	0.664	0.326	0.510	0.490
Faster R-CNN (MinneApple)	**0.712**	**0.938**	**0.426**	**0.574**
Faster R-CNN (COCO)	0.536	0.696	0.958	0.042

Table 2: Average Metrics per Image for each Method across the Test Set



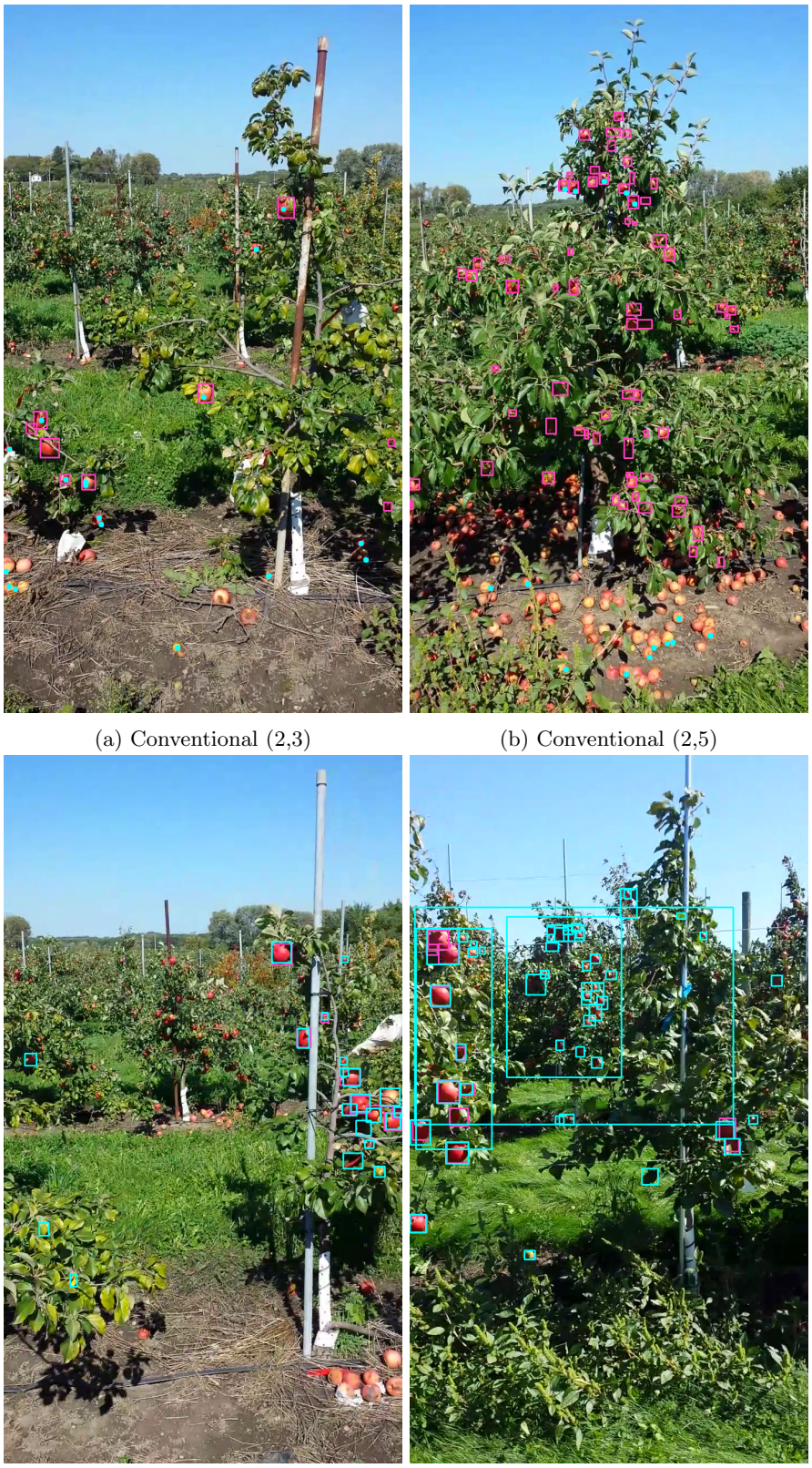
(a) Conventional (2,3)

(b) Conventional (2,5)

(c) Faster R-CNN (MinneApple)

(d) Faster R-CNN (COCO)

Figure 22: 90^{th} Percentile of Counting Accuracy for each Method



(a) Conventional (2,3)

(b) Conventional (2,5)

(c) Faster R-CNN (MinneApple)

(d) Faster R-CNN (COCO)

Figure 23: 10^{th} Percentile of Counting Accuracy for each Method

Base Detection Method	Border Size	Relative Err.	Counting Acc.
Faster R-CNN (MinneApple)	**10**	**0.109**	**0.891**
	20	0.138	0.862
	40	0.162	0.838
	80	0.205	0.795
Faster R-CNN (COCO)	**10**	0.087	0.913
	20	0.187	0.813
	40	0.275	0.725
	80	0.431	0.569

Table 3: Total Metrics for MOT across the Test Set

Base Detection Method	Border Size	Relative Err. (abs)	Counting Acc.
Faster R-CNN (MinneApple)	**10**	**0.326**	**0.674**
	20	0.356	0.648
	40	0.387	0.613
	80	0.439	0.561
Faster R-CNN (COCO)	**10**	0.655	0.345
	20	0.764	0.236
	40	0.870	0.130
	80	1.110	-0.110

Table 4: Average Metrics per Image for MOT across the Test Set

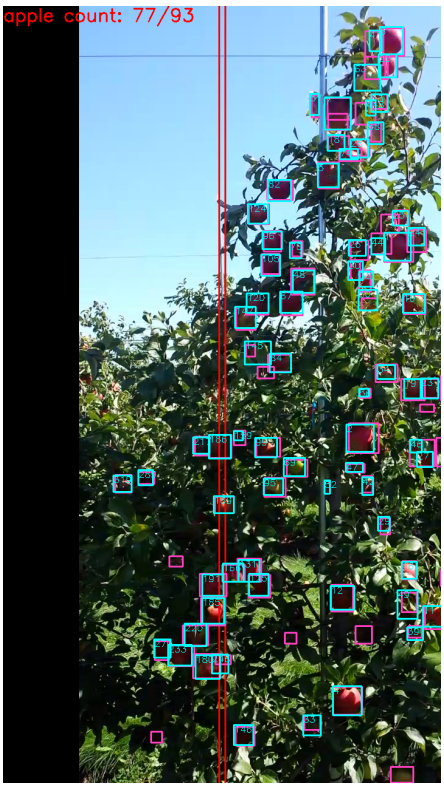


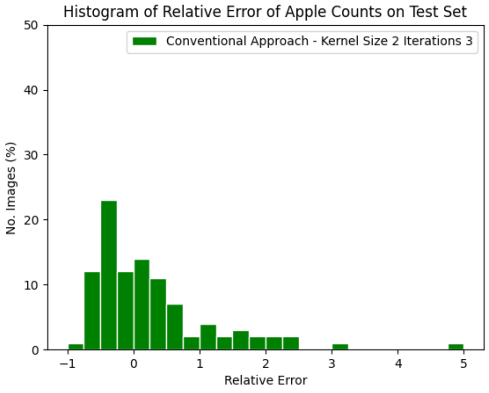
Figure 24: Screenshot of Qualitative Results of MOT - Base Detection Method Faster R-CNN (MinneApple); Border Size 10. Video Available at: `https://youtu.be/DiLGsKQz8q8`

6.2 Evaluation of Results

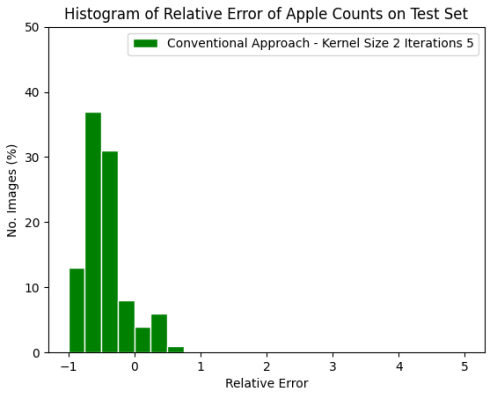
From the results in Tables 1 and 2 we see that the Machine Learning approach outperforms the Conventional approach in terms of Precision and Recall, where Faster R-CNN (MinneApple) achieved the highest scores in these metrics on average and in total. From this, we conclude that the Machine Learning approach has a better capability of detecting only relevant apples, as well as identifying a higher proportion of relevant apples in an image.

The highest counting accuracy across the total metrics was achieved by Conventional (2,3), despite significantly lower scores in Precision and Recall than Faster R-CNN (MinneApple). Since counting accuracy depends on the number of predictions regardless of TP or FP, the closeness of $\frac{TP}{FP}$ to 1 in Conventional (2,3) (Table 1 allows high FP to make up for high FN, resulting in lower relative error. While counting accuracy is higher, the solution is likely less generalisable to different datasets given low Precision and Recall.

The highest counting accuracy across the average metrics was achieved by Faster R-CNN (MinneApple). This corresponds with the approach's high precision and recall in Table 2, reflecting on its heightened ability to detect relevant apples. Training on MineApple outperformed the more general COCO dataset, implying that fine-tuning to the intended apple orchard before deployment will net a performance increase.

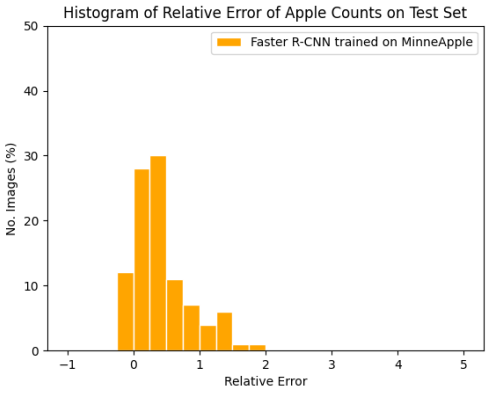


(a) Kernel size 2; Iterations 3

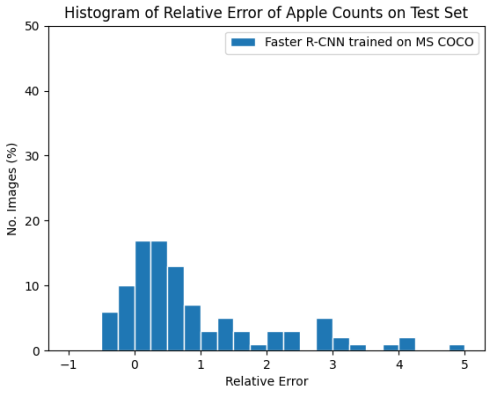


(b) Kernel size 2; Iterations 5

Figure 25: Histograms of Relative Error across the Test Set (Conventional Approach)

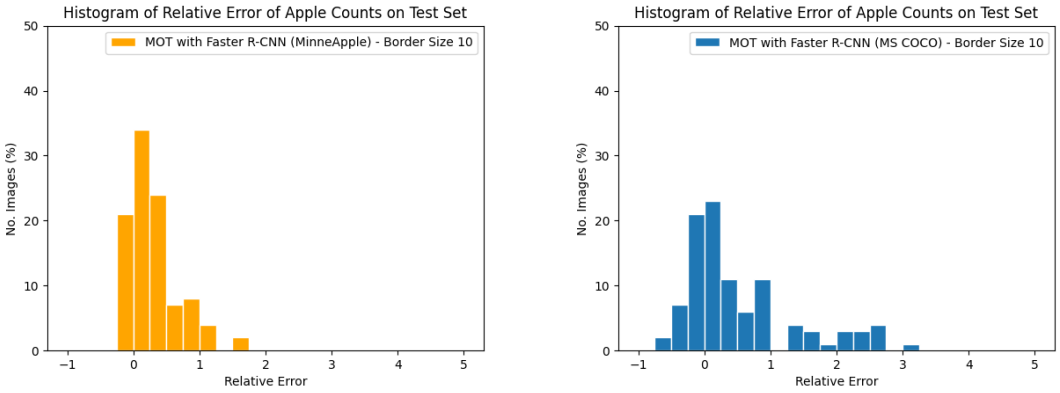


(a) Faster R-CNN trained on MinneApple



(b) Faster R-CNN trained on COCO

Figure 26: Histograms of Relative Error across the Test Set (Machine Learning Approach)



(a) Base detection method Faster R-CNN (MinneApple); Border Size 10

(b) Base detection method Faster R-CNN (COCO); Border Size 10

Figure 27: Histograms of Relative Error across the Test Set (Multiple Object Tracking)

All methods show a decrease in counting accuracy from total metrics to average metrics. This can be explained by observing both the quantitative and qualitative results. In Figures 25 and 26, a number of images exceed 1 in relative error across all methods evaluated on the test set. Qualitatively, Figure 23 shows the 10^{th} percentile of relative error in each method, with Figures 23a, 23c and 23d showing sparsely populated ground truths and a high number of FPs or relevant apples not labelled in the dataset. Such images are susceptible to high error as FPs have more weight, however do not contribute significantly to total metrics due to a sparse count of apples.

Qualitatively, we show from Figure 22 that all methods tend to perform better on densely populated images. We note from Figures 22 and 23 that Faster R-CNN (MinneApple) appears to predict more accurate bounding boxes than Faster R-CNN (COCO), however the latter is able to generalise to apples in the background not labelled in the dataset. Figure 23b highlights a disadvantage of conventional methods in that thresholding often does not generalise well to different lighting conditions, where many shaded apples are not detected.

Comparing Tables 3 and 4 to Tables 1 and 2, we see that both Faster R-CNN models exhibit higher accuracy in total and average metrics when MOT is applied to moving images for border sizes 40 and up. Comparing Figures 26 and 27, we observe that MOT introduces a bias away from positive error across the dataset, which likely corresponds to improvements seen as border size reduces in Tables 3 and 4. The ability of MOT to perform apple detection in moving images increases its versatility, as it is suited to tallying apples without repetition across many images captured by a mobile robot.

7 Conclusion and Future Works

Whilst there were some inherent challenges in completing this project: such as finding datasets specific to our problem, selecting the correct kernels for use in the conventional approach, and tuning the machine learning approaches for optimal function; We were able to successfully implement and evaluate both a conventional and machine learning approach for the task of counting apples in an orchard environment. In addition to this implemented an evaluation script was created to give accurate information about the precision, accuracy, and other rel event metrics to allow for direct comparison of the apple counting methods undertaken.

Our results showed that the machine learning approach was significantly more precise at identifying apples, as well as being more transferable to apples in different context environments (rather than being overfit to the training dataset). The conventional method was more accurate in counting apples over both the validation and testing dataset, however this was due to random predictions being drawn from noise in the image rather high precision in the predictions being made.

As mentioned in the previous sections, a possible extension of this work would be through the deployment of an MOT for identifications and classification as described in the methodology. By doing this we could employ a mobile robot to more efficiently count ripe apples by recording a video as it moves through the environment, rather than forcing it to stop and photograph individual trees (whilst simultaneously trying not to photograph the same apple more than once).

To further build on this we could apply the fuji 3D dataset, consisting of 3D models compiled from a large set of photos. This could make real world detection more viable, as a theoretical 'detection robot' would be able to identify an apple from multiple angles, decreasing potential duplicate counts and therefore increasing task accuracy. An alternative machine learning architecture may require investigating in order to allow the training of model capable of object detection at a rate feasible for this application.

References

Behera, S.K., R. A. . S. P. (2021), 'Fruits yield estimation using faster r-cnn with miou.', *Multimed Tools Appl 80* .
URL: *https://doi.org/10.1007/s11042-021-10704-7*

Collaborators, C. (2020), 'Coco - common objects in context'.
URL: *https://cocodataset.org/home*

Du, L., Zhang, R. & Wang, X. (2020), 'Overview of two-stage object detection algorithms', *Journal of Physics: Conference Series* **1544**(1), 012033.
URL: *https://doi.org/10.1088/1742-6596/1544/1/012033*

et al., W. Z. (2018), 'Broad-leaf weed detection in pasture', *IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)* .

Fu, L., F. Y. W. J. e. a. (2021), 'Fast and accurate detection of kiwifruit in orchard using improved yolov3-tiny model.', *Precision Agric 22* p. 754–776.
URL: *https://doi.org/10.1007/s11119-020-09754-y*

Gandhi, R. (n.d.), 'R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms'.
URL: *https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e*

Gené-Mola, J., Sanz-Cortiella, R., Rosell-Polo, J. R., Morros, J.-R., Ruiz-Hidalgo, J., Vilaplana, V. & Gregorio, E. (2020), 'Fuji-sfm dataset: A collection of annotated images and point clouds for fuji apple detection and location using structure-from-motion photogrammetry', *Data in Brief* **30**, 105591.
URL: *https://www.sciencedirect.com/science/article/pii/S2352340920304856*

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2013), 'Rich feature hierarchies for accurate object detection and semantic segmentation', *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* .

Gupta, A. (n.d.), 'A comprehensive guide on deep learning optimizers'.
URL: *https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/*

Häni, N., Roy, P. & Isler, V. (2020), 'Minneapple: A benchmark dataset for apple detection and segmentation', *IEEE Robotics and Automation Letters* **5**(2), 852–858.

Kuznetsova, A., M.-T. S. V. (2020), 'Using yolov3 algorithm with pre- and post-processing for apple detection in fruit-harvesting robot.', *Agronomy* .
URL: *https://doi.org/10.3390/agronomy10071016*

Li, L. (2022), 'Time-of-flight camera - an introduction— mouser'.
URL: *https://www.mouser.co.uk/applications/time-of-flight-robotics/*

Pytorch (n.d.*a*), 'Pytorch.org. 2022. torch.utils.data — pytorch 1.10.1 documentation'.
URL: *https://pytorch.org/docs/stable/data.html, addendum =*

Pytorch (n.d.*b*), 'Stop wasting time with pytorch datasets!'.
URL: *https://towardsdatascience.com/stop-wasting-time-with-pytorch-datasets-17cac2c22fa8*

Redmon, J. & Farhadi, A. (2018), 'Yolov3: An incremental improvement', *ArXiv* **abs/1804.02767**.

Ren, S., He, K., Girshick, R. & Sun, J. (2015), Faster r-cnn: Towards real-time object detection with region proposal networks, *in* C. Cortes, N. Lawrence, D. Lee, M. Sugiyama & R. Garnett, eds, 'Advances in Neural Information Processing Systems', Vol. 28, Curran Associates, Inc.
URL: *https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf*

Rosebrock, A. (2016), 'Intersection over union (iou) for object detection'.
URL: *https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/*

Shri V, R. (n.d.), '10 pytorch transformations you need to know'.
URL: *https://www.analyticsvidhya.com/blog/2021/04/10-pytorch-transformations-you-need-to-know/*

Zhenwei He, L. Z. (2019), 'Multi-adversarial faster-rcnn for unrestricted object detection', *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* .
URL: *https://openaccess.thecvf.com/content*$_{ICCV}$2019/*html*/$He_{Multi-Adversarial}{}_{Faster-RCNN}{}_{for}{}_{Unrestricted}{}_{Object}{}_{Detection}{}_{ICCV}2019_{paper.html}$

Zhu, N., Wang, G., Yang, G. & Dai, W. (2009), A fast 2d otsu thresholding algorithm based on improved histogram, *in* '2009 Chinese Conference on Pattern-https://www.overleaf.com/project/61dc4b634e330265205583de Recognition', pp. 1–5.

Appendices

A Resources and Code

- Project GitHub Repository: `https://github.com/joejeffcock-pg/ufmfrr_cw1`
- Faster R-CNN Weights: `https://uweacuk-my.sharepoint.com/:f:/g/personal/joe2_jeffcock_live_uwe_ac_uk/EjxKRK_wPqFDlJi9lg-S1OYBZvjjNBPkBOjHUDfxoKuGiw?e=xhkUTl`
- Video of MOT: `https://youtu.be/DiLGsKQz8q8`