Object Detection and Size Determination of Pineapple Fruit at a Juicing Factory

Minor Dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Data Science

Department of Statistical Sciences

University of Cape Town

Jessica Harris

July 2021

Supervisor: Dr Şebnem Er

Abstract

The aim of this thesis is to develop a method for determining pineapple fruit size from images. This was achieved by first detecting pineapples in each image using Mask Region-based Convolutional Neural Network (Mask R-CNN) and then extracting the pixel diameter and length measurements, and the projected areas, from the detected mask outputs. Various Mask R-CNNs were considered for the task of pineapple detection. The best-performing detector made use of MS COCO starting weights, a ResNet50 CNN backbone, and horizontal flipping data augmentation during the training process. This model (Model 4: COCO_Fliplr_Res50) achieved an average precision of 91.4% on the validation set and an average precision of 90.1% on the test set, and was used to predict masks for an unseen dataset containing images of pre-measured pineapples. The distributions of measurements extracted from the detected masks were compared to those of the manual measurements using two-sample Z-tests and Kolmogorov–Smirnov (KS) tests. There was sufficient similarity between the distributions, and it was therefore established that the reported method is appropriate for pineapple size determination in this context. All the data and code is available in a GitHub repository for reproducible research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives a brief overview of the need for representative fruit size data in the Eastern Cape pineapple juicing industry, outlines the research objectives, and describes the structure of this work.

1.1 Background

In the Eastern Cape of South Africa, the Smooth Cayenne pineapple cultivar is primarily grown for juicing purposes, with the juice being sold as a concentrated product with a prescribed sugar content. Fruit can be harvested from a pineapple plant twice in a five-year period, but planting is staggered to ensure the fruit can be harvested for at least 10 months of the year, from March to mid-December. In the context of the juicing industry, pineapples that are most suitable for processing are characterised by a high sugar content and a low ratio of peel to fruit. This work focuses on the latter feature.

Pineapples of all sizes are accepted at the juicing factory. However, a comprehensive study on the relationship between fruit size and juice yields has not been undertaken due to a lack of representative fruit size data. Currently, size measurements are acquired by taking several pineapples from each 30-ton truck load and physically cutting and measuring each fruit to obtain its diameter, length, and weight. As this is a destructive process, the number of pineapples measured in this way has been kept quite small to avoid negatively impacting the factory's output. Furthermore, this also means that the sample size has been too small to be truly representative of the population.

Theoretically, larger pineapples should have a higher juice yield because a smaller proportion of the fruit is comprised of peel, which is removed in the first step of processing. Once the relationship between size and yield has been fully established, the factory will be better positioned to incentivise the delivery of the most appropriately sized fruit. That is, instead of paying growers solely based on the weight of fruit delivered, a factor relating to fruit size could also be included in the pricing scheme. However, if payment is to be affected by fruit size, it is imperative that the fruit sampled for sizing is representative of all fruit delivered. Hence, this project aims to develop a non-destructive size determination approach that will enable future developments in yield- and revenue-optimisation.

1.2 Research objectives

There is a growing use of computer vision in the fields of agriculture and Agri-processing. The main goal of this research is to demonstrate how an object detector based on convolutional neural networks (CNNs) can be utilized in the non-destructive size determination of pineapples. In order to achieve this, this project aims to:

- review previous methods used in object detection and image segmentation, particularly in fruit industries,
- use object detection algorithms to identify fruit entering the factory,
- extract the size of the detected pineapples and determine whether the detected sizes are a true representation of the actual measurements acquired by traditional methods.

1.3 Structure of dissertation

This document comprises seven chapters, including the current introductory chapter. Chapter 2 gives some background on size determination in fruit industries, describing the need for size determination and exploring different approaches. The topic of fruit detection is also addressed briefly. Chapter 3 then goes on to describe the theory of convolutional neural networks, while Chapter 4 describes the modifications made to CNNs that allow them to be used for object detection. Chapter 5, Methodology, details the data collection process and provides further information on the preparation and annotation of the data set before detailing the practical implementation of the computer vision models for pineapple detection and size determination. It gives details regarding the software implementation, network architectures and the size determination approach employed in this work. Chapter 6 then presents the results and compares the performance of the different models employed. The validity of the size determination approach is also interrogated. Finally, Chapter 7 summarises the findings of the research, outlines the limitations of the work, and identifies areas for future work.

Chapter 2

Size determination and object detection of fruits

This first section of this chapter explains the interest in size determination in fruit industries and explores various reasons why it is necessary. Section 2.2 then goes on to discuss factors affecting fruit size before Section 2.3 reviews electronic systems used in the determination of fruit and vegetable size. Finally, Section 2.4 reports on several methods that have been used for fruit detection.

2.1 Interest in size determination

Fruit size is described by weight, or by some dimensional parameter such as length, diameter, volume, circumference, projected area, or some combination of these [1, 2]. In-field size determination allows for yield prediction [3, 4] and can be used as a parameter in predicting optimum harvest time [2]. However, there is also a need for size determination postharvest. Postharvest size determination is important in several contexts, such as sorting fresh market fruits into size groups for packaging purposes, and for assigning prices [2]. Uniform size is required by several regulatory boards for fresh pineapple sales, for example [5, 6]. It has been shown that homogeneity, as well as appearance of the packaged fruit, can have a significant effect on a consumer decision in fresh fruit sales [1]. Postharvest size determination is also used to sort fruit into size groups prior to processing. Extractors in citrus juice plants, for example, are generally designed for a given fruit size [2].

In the context of this work, size determination is not linked to any sorting activity, as

all fruits are homogenised in the juicing process. However, the first processing step involves peeling of the pineapples and it should be noted that the peelers remove a set width of peel from each pineapple, regardless of fruit size. Theoretically, this means that there will be more wastage with small fruits, as a larger percentage of the fruit is removed by the peeler. Hence, larger pineapple fruits with a lower ratio of peel to flesh are expected to have a higher juice yield per kilogram of fruit. In the context of this work, size determination is a necessary step towards being able to quantify the relationship between fruit size and juice yield.

2.2 Factors impacting fruit size

Collecting representative fruit size data is beneficial not only to the juicing factory but to the pineapple growers as well. Access to representative fruit size data means that farmers will be able to better assess their crop management practises. Fruit size is influenced by genetics, growing conditions, and the stage of growth [7]. The growing conditions include a wide variety of factors including soil type and mineral nutrition, plant population or planting density, water availability, pest management, as well as growing temperature and irradiance [7]. It has been shown that pineapple cultivars show significant variation in their plant growth and fruit size when grown in different environments [8].

Soil type, soil fertility and water content are factors affecting fruit size [9]. For example, the pH of the soil, which indicates its acidity or alkalinity, determines whether the minerals in the soil are available to the plants for uptake [7]. Fertilization is one farming practise that can be used to improve the quality of the soil, making more nutrients available for plant growth and fruit production [7]. In pineapple crops, nitrogen has been shown to be essential to the increase of fruit size and total yield [10]. Pineapples can be grown in areas of low to moderate rainfall (500–2000 mm/year) without irrigation [11] and tend to be grown on ridges to prevent standing water in the root region, which can hamper plant growth.

The spacing of pineapple plants, the planting density, has been shown to affect the average fruit weight and yield per unit area [8]. Although high planting density tends to result in an increase in total yield, it also leads to a decrease in the average fruit size [8, 10, 12]. Fruit weight is significantly correlated with the mean irradiance between planting and harvest [8]. Hence, if plants receive less sunlight due to mutual shading in areas of high planting density, they will tend to produce smaller fruit [8]. The orientation of a field will also affect the mean irradiance: plants grown on a north-facing slope (in the Southern hemisphere) will receive more sunlight and will tend to produce larger fruit.

The presence of insects and mites, diseases, and weeds may reduce fruit quality and impact fruit size [7]. Pests may infect plants with diseases resulting in smaller fruit [11]. The first production cycle of pineapples is referred to as the plant crop. After the plant crop has been harvested, vegetative suckers are produced by the parent plant. The crop that results from these suckers is referred to as the ratoon crop. Ratoon crop cycles are only possible if the effects of pests such as mealybugs and nematodes are limited [11]. Furthermore, weeds can cause serious crop losses. In Guinea, an unweeded pineapple crop was shown to produce fruit with an average size equal to half of that produced by a similar crop with good weed control [8].

2.3 Systems for determination of fruit size

Electronic sizers have now almost completely replaced mechanical sizers [2]. Electronic systems for determination of fruit size may be based on measuring volume of gap between fruit and an outer casing, time of flight (TOF) systems that measure the time for light to pass between the light source and the fruit, or on blocking of light, in which case a light source and photodiode receiver are placed on either side of conveyor [2]. In recent years, there has been an increased interest in the use of machine vision approaches in fruit industries [13]. This work will focus on the use of machine vision systems for fruit detection and measurement, particularly those making use of convolutional neural networks (CNNs).

Machine vision systems may either be two-dimensional or three-dimensional [2]. Two-dimensional machine vision systems typically use digital cameras to take images of fruits [14]. These images are then analysed, with the aim of extracting dimensional features such as width, although there are several two-dimensional size measures that can be considered. The typical axes used for fruit size and simple shape determination are the diameter and length [2]. Projected area has also been used for classification into size groups [2]. When projected area is used, it is necessary to know corresponding pixel size of the specific camera height in order to determine the projected area of the actual fruit. In an example from the citrus industry, once the boundary of a fruit was defined using edge detection, the fruit's centroid coordinates were calculated. The distance between each boundary pixel and the centroid was then calculated and the length of the fruit was found by identifying the maximum distance, while the width was found by passing through the centroid, perpendicular to the first axis. This algorithm has the advantage of being independent of fruit orientation [2].

Three-dimensional machine vision systems aim to address the lack of information about height or depth in two-dimensional images [14]. Three-dimensional techniques can either be

active or passive. Passive techniques rely on using 3D cues in images, such as shadows, to estimate the shape of an object. Another passive technique is stereovision, which involves using images of the same fruit from different perspectives [15]. With stereovision, it is necessary for the corresponding images to have matching points – points on the object that can be identified in both images. Identifying these matching points can be a time-intensive process. Active techniques, on the other hand, involve projecting energy such as laser light onto the surface of the object. If a plane of laser light is projected onto an object, the height can be obtained from the distortions in the light, based on triangulation [2].

Three-dimensional machine vision techniques employ what are known as range images, which are a collection of distance measurements between sensors and objects [2, 14]. In a range image, each pixel represents the distance from the reference frame to a visible point in the scene. An RGB-depth (RGB-D) camera can be used to collect both colour image and depth image simultaneously [14].

2.4 Convolutional neural networks (CNNs) in fruit image processing

Early fruit detection methods relied on static colour thresholds [16]. An example of this is the citrus counting algorithm [17], which consisted of the following steps: converting RGB image to HSV, thresholding, orange colour detection, noise removal, watershed segmentation, and counting [17]. The reliance of these methods on a colour differential was alleviated by using additional sensors, such as thermal or near infrared (NIR) [16]. While earlier computer vision approaches required manual selection of features and static colour thresholds [16, 18, 13], CNNs do not require manual feature selection and are able to learn complex and high dimensional features automatically – they only need to be given input images with associated labels [13, 19]. While the theory of CNNs is discussed in detail in Chapter 3, some examples of applications of CNNs in the agricultural sector are discussed below.

CNNs have been used in fruit image tasks that include classification, quality control and detection [13, 19]. Classification tasks may involve identifying fruits or weeds [20], while quality control tasks may involve identifying fruit defects or disease, for example [13, 20, 21]. Fruit detection tasks are particularly important in fruit counting for the purposes of yield predictions or robotic harvesting [19, 20].

In 2015, Faster R-CNN [22] became the state-of-the-art object-detection method [16]. Faster R-CNN has been used for detection of apples, mangoes and almonds in an orchard

setting [23], and of sweet pepper and rockmelon detection in a greenhouse environment [24]. Since then, there has been a marked increase in the number of articles based on CNNs for fruit image processing [13]. An advantage of object-detection-based counting is that it does not require additional counting algorithms. Instead, the detected instances are summed up to give the fruit count [16]. This was further extended in 2017, when Mask R-CNN [25] allowed for the identification of pixel-wise masks for each detected fruit in an image [13], allowing for separation between fruit and background. Mask R-CNN has been used to detect oranges [26] and strawberries [27] for automatic harvesting.

Chapter 3

Convolutional neural networks

The methods of fruit detection mentioned in Section 2.4 are based on convolutional neural networks (CNNs). This chapter aims to give an overview of CNNs and the mechanisms involved in each layer of the network. The Background section first introduces feature detection in images and then goes on to explain the convolution operation and the rationale for using it in deep learning applications. The different types of layers found in a convolutional neural network are then described in the Structure section, along with important concepts associated with each type of layer. Finally, the Learning section gives an overview of the learning algorithm used to find the weights of the trainable filters and discusses various optimisers that could be employed.

3.1 Background

This work deals with image inputs. As such, images and image features are first discussed to provide some background. Thereafter, the convolution operation, which is used to identify features in an image, is explained.

3.1.1 Images and features

Colour images stored in red, green, blue (RGB) format are order 3 tensors. They have height (H), width (W) and 3 channels for R, G and B, respectively. Each channel is a $H \times W$ matrix that contains the respective R, G or B values of all pixels [28]. This means that each pixel represents the colour at a point in the image [29]. Greyscale images, on the other hand, are

two-dimensional: they have height (H) and width (W) but only have one channel containing pixel values ranging from 0 (white) to 255 (black).

Image features may include brightness, colour or texture [30]. Another type of feature is an edge, which represents a local discontinuity in image brightness [30, 31]. Features may also be more complex and could comprise entire objects. In computer vision tasks, it is often useful to know where different features occur in an image. Convolution, which is discussed in detail in Section 3.1.2, can be used to construct feature maps, which indicate where certain features appear in the input image [32]. These feature maps provide useful information and can be converted to vector form and treated as nodes in a traditional neural network for the purposes of image classification [33], a process which is described in more detail in Section 3.2.3.

3.1.2 The convolution operation

Convolutional neural networks (CNNs) are neural networks that use a convolution operation instead of matrix multiplication in at least one layer [32]. CNNs works particularly well for processing data with a grid-like topology, including images, time-series, audio and text data [32], where the same feature needs to be detected in multiple places in the grid [34]. As this work deals exclusively with image data, image-based examples will be used to illustrate convolution in this chapter.

Convolution is a mathematical operation denoted with an asterisk ($*$). The input is usually a multidimensional array of data, and the filter is usually a multidimensional array of parameters (weights) that are adapted by the learning algorithm [32]. Convolving a two-dimensional input image (I) with a two-dimensional filter (F) gives the following output [29, 35]:

$$O(i,j) = (I * F)(i,j) = \sum_m \sum_n I(m,n)F(i-m, j-n) \tag{3.1}$$

where i and j are the convolution output (O) indices and m and n are the input image (I) indices. Discrete convolution is equivalent to performing a dot product between the filter weights (F) and the image values underneath the filter, centred at (i,j), to produce an (i,j) value for O. Dot products measure similarity; a maximal value in the feature map is achieved when the intensity pattern in the image's pixel values match the weights in the filter. The output is also referred to as a feature map, as it shows where certain features appear in the

input [32]. As convolution is commutative, Eq. 3.1 can equivalently be written as [32]:

$$O(i,j) = (I * F)(i,j) = \sum_m \sum_n I(i-m, j-n)F(m,n) \tag{3.2}$$

Equation 3.2 is usually preferred to Eq. 3.1 in machine learning implementations, as there is less variation in the range of valid values of m and n [32]. In this case, the filter has been flipped, along both the x- and y- axis, with respect to the input image. In practise, however, many machine learning libraries use cross-correlation (Eq. 3.3), which is the same as convolution but without flipping the filter, although it is still referred to as a convolution operation.

$$O(i,j) = (I * F)(i,j) = \sum_m \sum_n I(i+m, j+n)F(m,n) \tag{3.3}$$

Fig. 3.1 gives an example of the discrete convolution operation, using Eq. 3.3. To apply a filter to an input image, the filter is placed on the image and the filter weights are multiplied with the underlying image pixels. These are then summed, generating a single pixel value for the output. The filter is then moved along to every valid position within the image and the process is repeated at each step, generating a feature map. The region that is covered by the filter at each instance of the convolution operation is called the receptive field [32]. While filter weights can be hand-picked, they can also be treated as a matrix of parameters, which can be learnt using back propagation. Filters in earlier layers learn simple features, such as edge detectors, while filters in deeper layers learn increasingly complex patterns and features [36, 37]. In earlier levels, the receptive field is comprised of a small collection of pixels, whereas deeper layers have larger receptive fields and can pick up on patterns and features that extend over greater spatial regions.

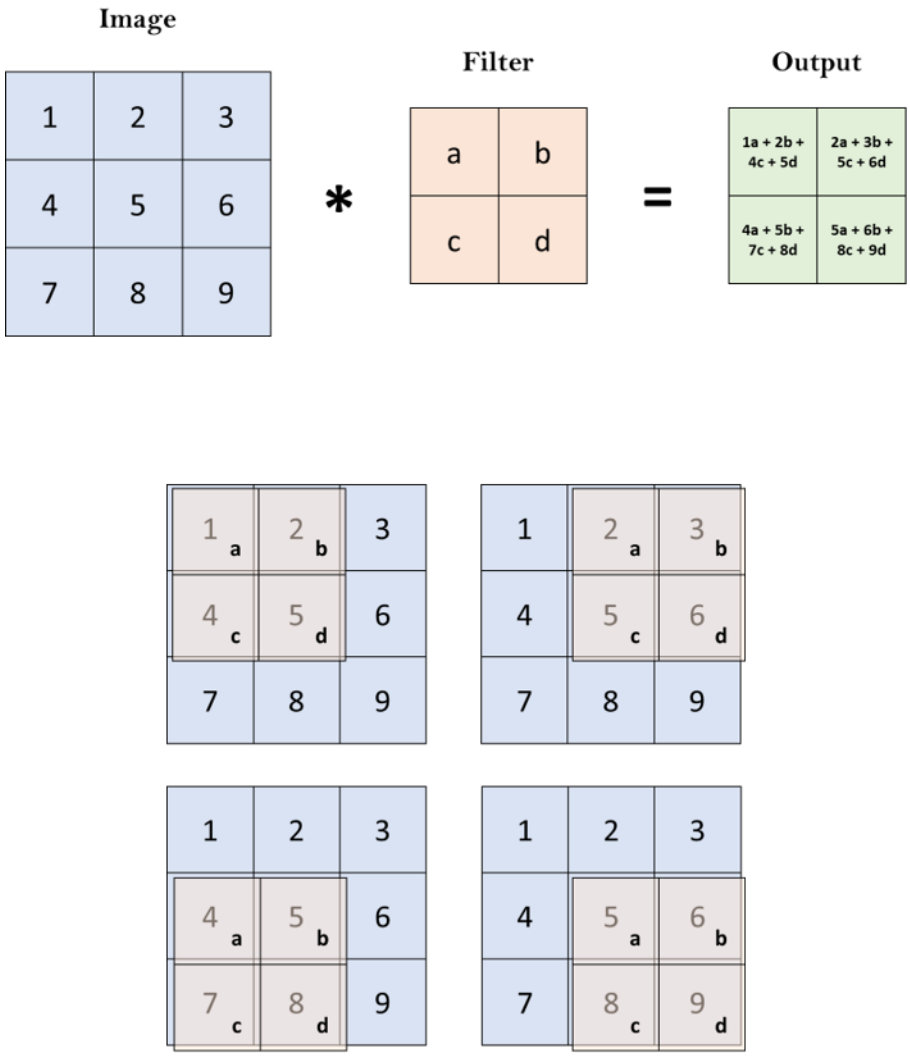

Figure 3.1: *Illustration of convolution operation in 2-D. Top shows a* 3×3 *image input and a* 2×2 *filter, together with the feature map output. Bottom shows the filter being shifted to all positions where the filter lies entirely within the image. In each position, the product is computed between the filter and input and these products are then summed together to get a single number, as shown in the output above.*

Convolution leverages sparse connections and parameter sharing [32]. Sparse connectivity (Fig. 3.2) describes the fact that each output value only depends on a small number of inputs. Parameter sharing refers to the fact that a filter can be used across all spatial locations in an image [28]: a feature detector that is useful in one part of an image is likely to be useful in another area in the image. This allows for a single filter to be used across the entire image, detecting a given feature at each position. This means that the number of parameters does not depend on the size of the image. Instead, the number of weights (parameters) that must be learnt is determined by the size and number of filters chosen. These two mechanisms greatly decrease the number of network parameters, allowing the network to be trained with fewer training images and reducing the risk of overfitting [32].

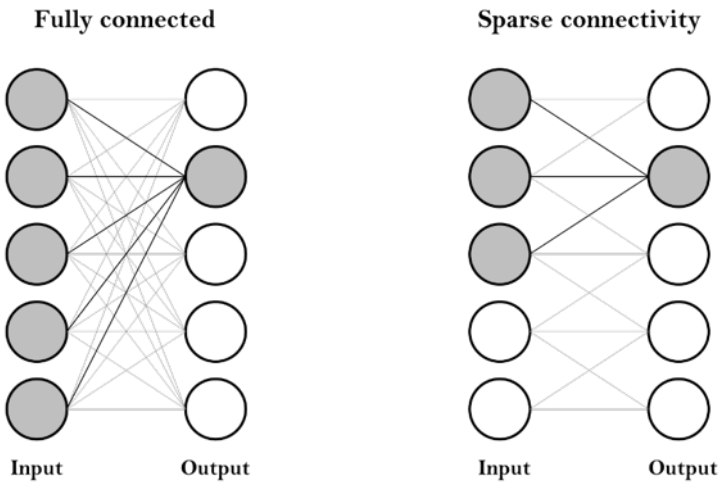

Figure 3.2: *Fully-connected layers, where the output is formed by matrix multiplication (left) compared to sparse connectivity, where the output is formed by convolution (right). In each case, one output neuron is highlighted in grey. The neurons in the input layer that affect the highlighted neuron are known as the receptive field and are also highlighted. In the case of the fully-connected layer (left), all neurons in the input layer affect the highlighted output neuron. When the output is formed by convolution with a filter of width 3 (right), however, only 3 input neurons affect the highlighted output neuron.*

3.2 Structure

In a CNN, an input undergoes a number of processes that are performed sequentially. Each processing step is referred to as a layer [28]. CNNs typically have three types of layers: convolutional, pooling and fully connected (Fig. 3.3). The convolutional and pooling layers together perform feature extraction, while the fully connected layer is responsible for classification. In terms of notation, there are several approaches to numbering layers. One approach is to only report the number of layers that have parameters. In that case, the convolutional and pooling layers shown in Fig. 3.3 would, together, make up Layer 1, as pooling layers do not have any parameters that need to be learnt. In this work, however, the convention of numbering each layer separately will be used.

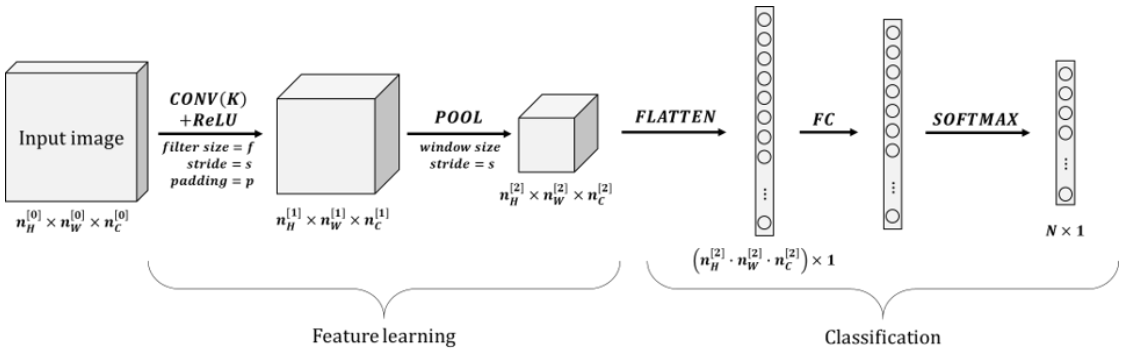

Figure 3.3: *Structure of a convolution neural network, showing the different types of layers and the dimensions of the data at each stage. In this example, the input image is convolved with K filters in the convolutional layer before applying a ReLU non-linear activation function. Thereafter, a pooling layer reduces the height and width dimensions of the representation while preserving the number of channels. The representation is then flattened into a vector before the fully connected layer. Finally, a softmax layer with N nodes is used to classify images into one of N categories.*

3.2.1 Convolutional layer

In a convolutional layer, the input is convolved with a trainable filter, represented by a matrix of weights, to give a feature map output, as shown in Fig. 3.1. The aim of the convolutional layer is to learn feature representations of the inputs [33]. Each neuron of a feature map is connected to a region of neighbouring neurons in the previous layer, referred to as the neuron's receptive field [33]. In each convolutional layer, several convolutions are typically run in parallel to extract multiple features from the input, as each filter learns only one feature. Hence, if there are K filters used in a convolutional layer, the output will have K

channels (Fig. 3.4). The number of channels in each filter must be equal to the number of channels in the previous layer. However, the height and width dimensions of the filter, and the number of filters to use are hyperparameters that need to be chosen by the user.

Colour images, which are often represented using RGB colour models, have 3 channels for red, blue and green, respectively. Figure 3.4 shows an example of a $5 \times 5 \times 3$ RGB image being convolved with two 3×3 filters. By default, the number of channels in each filter will be equal to the number of channels in the input image. In general, convolution of an image of dimension $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$ using K filters of size $f_H \times f_W \times n_C^{[l-1]}$, where f_H and f_W are the height and width of the filters, which are smaller than that of the image, and $n_C^{[l-1]}$ is the number of channels in the input image. The dimension of the output is given by Eq. 3.4 [28]:

$$(n_H^{[l-1]} - f_H + 1) \times (n_W^{[l-1]} - f_W + 1) \times K \tag{3.4}$$

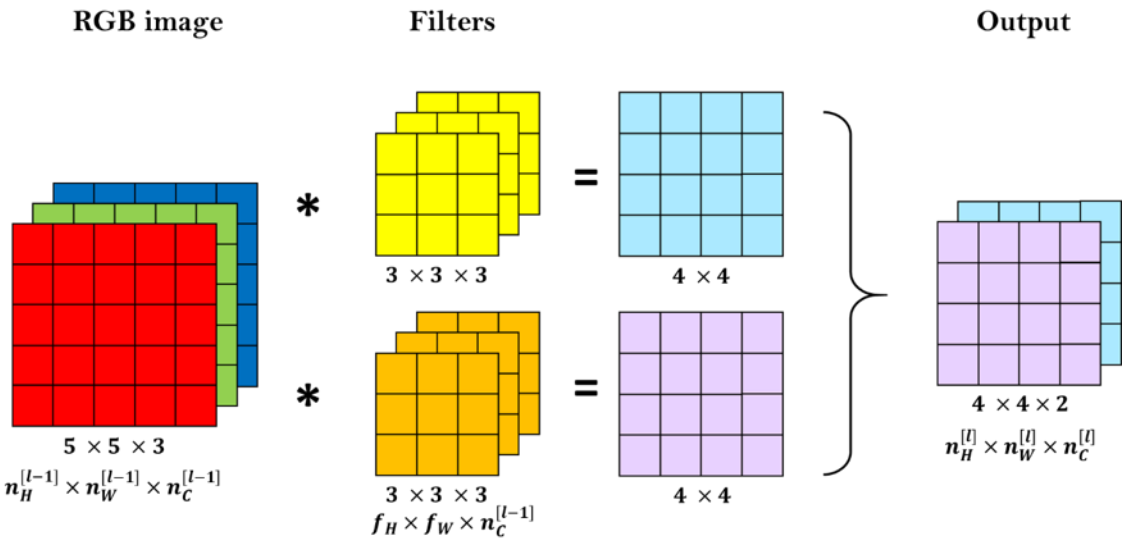

Figure 3.4: *Convolution operation for a multi-channel input image, with multiple filters. A nonlinear activation function, such as ReLU, would be applied to the output before being presented to any subsequent layers. Note that the number of channels in the output is equal to the number of filters used. The height and width of the output are reduced with respect to the input dimension.*

Padding

To prevent the width of output shrinking at each layer, a process known as "zero padding", which involves symmetrically adding zeros to the input, may be used (Fig. 3.5).

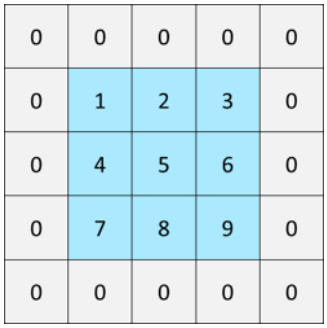

Figure 3.5: *Zero padding of a 2D input.*

Two special cases are "valid" convolution, where no padding is used, and "same" convolution, where just enough zero padding is added to keep the output size equal to the input size [32]. The dimension of the output given with p pixels of padding is given by Equation 3.5:

$$(n_H^{[l-1]} + 2p - f_H + 1) \times (n_W^{[l-1]} + 2p - f_W + 1) \times K \tag{3.5}$$

The number of pixels of padding required to yield an output that has the same size as the input ($n^{[l-1]}$), given a filter of size $f_H = f_W = f$, can be found by setting the size of the output equal to that of the input, and solving for p:

$$\begin{aligned} n^{[l-1]} + 2p - f + 1 &= n^{[l-1]} \\ 2p - f + 1 &= 0 \\ p &= \frac{(f-1)}{2} \end{aligned} \tag{3.6}$$

By convention, f is usually odd. This allows for symmetric padding to be used. Additionally, a filter of odd dimension has a pixel in the central position, which is convenient for describing the position of the filter. Padding is useful because it makes it possible to apply a convolutional layer without shrinking the height and width of the outputs, which is an important consideration when building deeper networks. If large filters are used in the absence of zero padding, the size of the output reduces rapidly, limiting the number of convolutional layers that can be implemented. This reduces the number of higher-level features that can be learnt, which will negatively affect the performance of the model [32]. Using small filters would reduce the rate at which the output size shrinks but this would also limit the perfor-

mance of the model, as it would not be able to learn any features that occur over larger areas of pixels [32]. Furthermore, padding helps us preserve more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

Stride

Stride is another hyperparameter in the convolutional layer. Stride, s, refers to the relative offset applied to the filter [38]. That is, the number of horizontal or vertical steps a filter takes when convolving an image input [32]. Convolving the filter at every possible spatial location corresponds to a stride of 1. If the stride is set to $s > 1$ then convolution is performed every s pixels, both horizontally and vertically [28]. In doing so, the overlap of receptive fields is reduced, as well as the spatial dimensions of the output. It is also possible to define a separate stride for each direction of motion [32]. When considering both stride and padding, the size of the output can be described by Equation 3.7:

$$\lfloor \frac{n_H^{[l-1]} + 2p_H - f_H}{s} + 1 \rfloor \times \lfloor \frac{n_W^{[l-1]} + 2p_W - f_W}{s} + 1 \rfloor \times K \tag{3.7}$$

The brackets $\lfloor \ \rfloor$ in Eq. 3.7 indicate the floor function. Hence, if the number given by is not an integer, it is necessary to round down to find the dimension of the output.

Activation functions

After convolution of an input image with a filter, the linear activations in the feature map are passed through a non-linear activation function. The non-linear activation is also known as the detector stage [32], as it is responsible for detecting the presence of the feature extracted by the filter. Typical activation functions are the sigmoid, hyperbolic tangent ($tanh$) and the rectified linear activation ($ReLU$) functions [33], given by Equations 3.8—3.10, respectively:

$$\sigma(x) = \frac{1}{(1+e^{-x})} = \frac{e^x}{(e^x+1)} \tag{3.8}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.9}$$

$$ReLU(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases} \tag{3.10}$$

The training time for CNNs with non-saturating $ReLU$ activation functions has been found to be much faster than those with saturating activations such as sigmoid or $tanh$, where the output is constrained by upper and lower limits (Fig 3.6) [39]. The output, having passed through the non-linear activation, becomes the input to subsequent layers. For the purpose of training, it is essential that the non-linear activation functions are differentiable. $ReLU$ is differentiable at all points except zero. However, piecewise differentiation can be used, allowing it to be used it situations where a differentiable function is required. In this case, the derivative of the $ReLU$ function at $x > 0$ is one, while it takes a value of zero elsewhere.

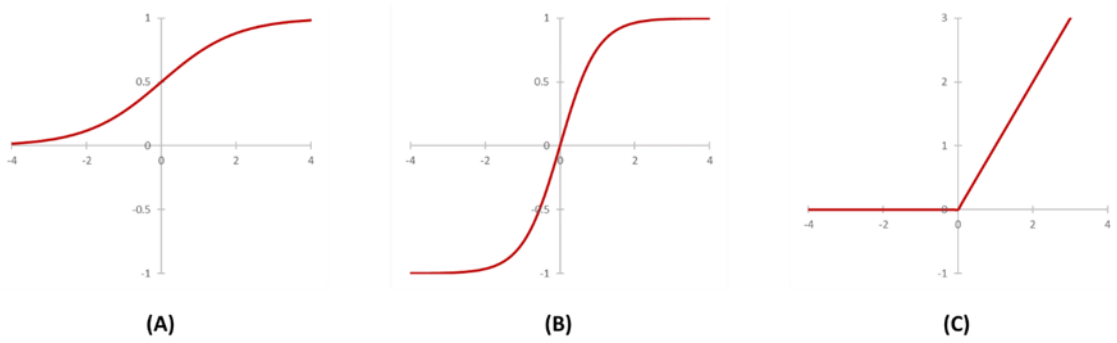

Figure 3.6: *Activation functions (A) sigmoid, (B) tanh, and (C) ReLU.*

3.2.2 Pooling layer

In a pooling layer, low-level features from the output of a convolutional layer are aggregated over a small neighbourhood defined by a window with pre-set size (Fig. 3.7) [40]. The purpose of a pooling layer is to reduce the spatial size of a layer, thereby decreasing the number of parameters that need to be optimised. This can speed up computation as well as prevent overfitting [40].

In pooling layers, the window size and stride are hyperparameters that must be defined. However, there are no parameters in a pooling layer that need to be learnt. If an input has multiple channels, the output will have the same number of channels, as the pooling operation is applied independently to each of the channels. Maximum pooling and average pooling are commonly used pooling functions (Fig. 3.7) [28]. The maximum pooling function applies a window function $u(x, y)$ to the input patch and computes the maximum in the neighbourhood, resulting in a feature map of lower resolution [40]. Only the maximum value within the receptive field is propagated to the next layer. Similarly, average pooling computes the average in each neighbourhood defined by the window function and propagate these average values to the next layer. Traditionally, non-overlapping pooling regions are used (as shown in Fig 3.7). However, if the stride is set to a value less than the window size, overlapping pooling is obtained [39].

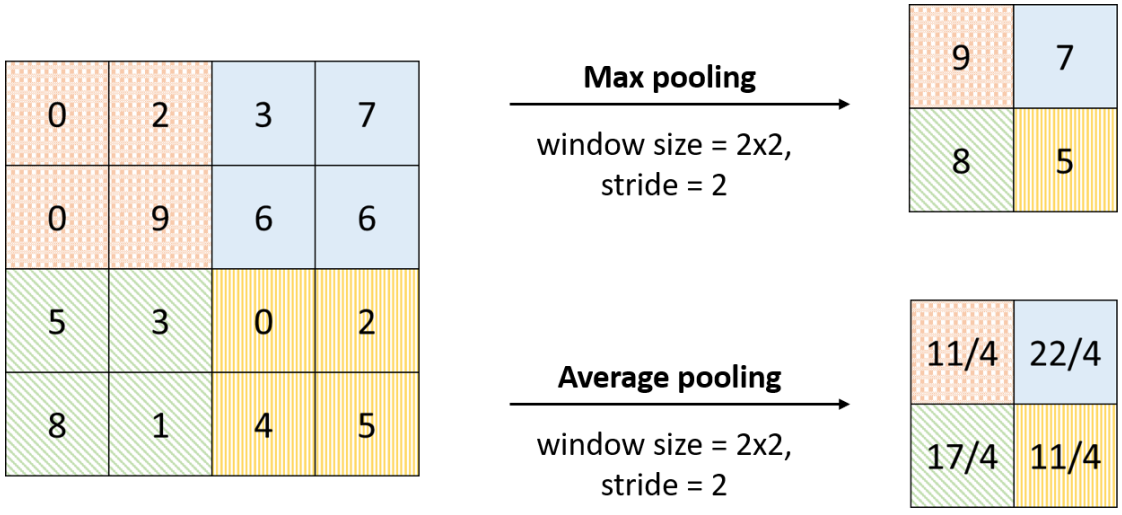

Figure 3.7: *Illustration of maximum and average pooling functions using non-overlapping windows with window size of* 2×2 *and stride of 2. Note that the dimensions of the feature map have been reduced from* 4×4 *to* 2×2.

3.2.3 Fully-connected layer

In the fully connected layer, the output from the previous layer is vectorised and concatenated (flattened (Fig 3.3)), giving the feature map in vector form. The elements in the vector are then treated as nodes in one layer that are connected to every node in another layer and matrix multiplication is applied as per traditional neural networks [33]. The last layer of CNNs is an output layer. In the case of classification tasks, where a single label is given to indicate the subject of entire input image, the softmax function (Eq. 3.11) is commonly used [39, 33]. Softmax produces a distribution over the K class labels [39].

$$f_k(x) = \frac{e^{x_k}}{\sum_{i=1}^{N} e^{x_i}}, k = 1, 2, 3...K \tag{3.11}$$

The fully-connected layer can be adapted to accommodate cases where the localisation of an object within an image is required [41]. The softmax classifier layer can be replaced with a regression network and trained to predict bounding boxes to locate objects within an image. The classifier and regressor networks can then be run simultaneously to generate bounding box predictions [42]. This is known as object detection, which will be covered in more detail in Chapter 4.

3.3 Learning

The weights in the trainable filters — the parameters — need to be learnt. Model training and optimisation are discussed in the Algorithm overview section. Subsequently, some relevant terminology is clarified. Finally, a selection of optimisers is presented.

3.3.1 Algorithm overview

Backpropagation is used to learn model parameters that fit the data well. The backpropagation algorithm is a numerical method used extensively in neural networks [43]. It comprises three processes: forward propagation, backward propagation, and parameter update (Algorithm 1). In practise, the model parameters are initialised and then training data is presented to the network, generating a predicted output. In the case of a classification problem, this would be a predicted class output. The predicted outcome is then evaluated using an appropriate loss function that describes how the predicted output differs from the actual class labels. In the back-propagation step, the chain rule is used to compute the derivative of the loss function with respect to each of the model parameters one layer at a time, beginning at the last layer [32]. The derivative of the loss function with respect to each parameter is the gradient. The weights are then updated using the method of gradient descent, or some variation thereof (discussed further in Section 3.3.3), and the loss is recalculated. This is repeated until some iteration threshold ($threshold_1$) is reached, or until the change in loss is negligible (i.e. smaller than $threshold_2$). The aim is to find the combination of weights that minimises the loss function [43]. As this method requires gradient computations at each step, the error function should be continuous and differentiable [43]. It is therefore essential that the non-linear activations used in the network be differentiable.

Algorithm 1: Backpropagation algorithm

Input: Training data

Result: Optimised model parameters

Initialise model parameters

while $iteration > threshold_1$ *or* $\Delta error \leq threshold_2$ **do**

    Forward propagation

    Backward propagation

    Update model parameters

end

3.3.2 Terminology

There is some terminology that is essential to the discussion of model training. Optimisation is the process of minimising or maximising a function $f(x)$ with respect to its parameter, x. The function is referred to as the objective function. A training epoch refers to one sweep through the entire training set [44]. The batch size, b, is defined as the number of examples presented to the model in a single cycle. It is a hyperparameter that controls the number of training samples to work through before the model's internal parameters are updated [44]. Finally, an iteration describes the number of times a batch of b examples is passed through the model.

3.3.3 Optimisers

Gradient descent

Gradient descent is a popular algorithm used in the optimisation of neural networks [45]. The derivative of the objective function $f(x)$ with respect to x gives the gradient of $f(x)$ at the given value of x. It is useful when minimising a function as it provides information as to how changing the value of x will affect the value of $f(x)$ [32]. When the objective function has multiple parameters, partial derivatives are used [32]. Gradient descent minimises the objective function by adjusting the parameters in the opposite direction of the gradient of the objective function with respect to the parameters [45]. The learning rate is a positive scalar value that determines the size of the steps taken in the direction of the negative gradient, to reach a local minimum [45, 32].

There are three main variants to gradient descent, relating to how much data is used to compute the objective function. These are referred to as batch, stochastic and mini-batch gradient descent [45]. Batch gradient descent computes the gradient of the cost function with respect to the parameters for the full training set [45, 32]. That is, the batch size is equal to the number of observations in the dataset: $b = N$. As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow, especially for large training sets [45, 32]. Stochastic gradient descent (SGD), on the other hand, performs a parameter update for each training example, making it much faster and allowing for the approach to be used online. However, the parameter updates are highly variant, making SGD prone to overshooting the minimum [45]. In mini-batch training, parameter updates are performed after every mini-batch of b training examples, where $1 < b < N$. This reduces the variance of the parameter updates, allowing for more stable convergence [45].

Momentum

Stochastic gradient descent has trouble navigating areas where the surface curves more steeply in one dimension than another (also referred to as "ravines") [45]. Momentum is a method that helps accelerate convergence to a local minimum [46], especially where there is high curvature or noisy gradients [32]. The momentum algorithm works by replacing the gradients in SGD with exponentially weighted moving averages of all the previous gradients when calculating the new values for the parameter weights [32]. That is, the weight change can be expressed in terms of the previous weight change and the current gradient [47].

Root mean square propagation (RMSProp)

Root mean square propagation (RMSProp) [47] involves dividing the learning rate by the square root of the exponentially weighted moving average of the gradient [32]. This method allows the learning rate to be different for each of the parameters. The base learning rate is adjusted using the square of the gradients, which is a measure of how volatile the gradients are. In this approach, highly volatile gradients have a reduced learning rate, reducing oscillation and allowing for quicker convergence. RMSProp has been successfully used for optimisation of deep neural networks [32].

Adaptive moment estimation (Adam)

Adam (adaptive moment estimation) is another optimisation algorithm that makes use of adaptive learning rates [32]. Each parameter's individual adaptive learning rate is calculated from estimates of first and second moments of the gradients [32, 48]. Adam stores an exponentially decaying average of past squared gradients, as per RMSProp, as well as the exponentially decaying average of past gradients, as per momentum [45].

In this chapter, the theory of CNNs has been discussed in the context of classification tasks. However, CNNs can be used in a variety of different computer vision tasks including object detection, which is the subject of Chapter 4.

Chapter 4

Object detection

The CNNs discussed in Chapter 3 may be adapted and used for tasks other than classification. One such task is object detection, which will be the subject of this chapter. The Background section describes what object detection is and notes the challenges that are specific to it. The differences between object detection and instance segmentation are also discussed. As it is important to be able to quantify how well an object detector is performing, the Evaluation criteria section describes several methods used to assess detectors. In the Detection frameworks section, the naïve sliding window approach to object detection is described as an introduction to object detectors. Thereafter, a series of region-based approaches are discussed. Finally, Mask R-CNN, which is the framework that will be used in this work, is detailed.

4.1 Background

The goal of object detection is to locate all object instances in an image from a list of predefined classes [49]. Hence, the problem involves both predicting the category as well as drawing a bounding box around each object in an image. A bounding box is a set of coordinates (x, y, w, h) describing the rectangular border that fully encloses an object and indicates its location within an image. The x and y values give the x- and y-coordinates of the centre of the bounding box, while w and h give the width and height of the box, respectively (Fig. 4.1). By convention, the top left corner of the image has (x, y) coordinates of $(0, 0)$.

Figure 4.1: *Given an image containing an object of interest (left), a bounding box can be drawn around the object (right). The bounding box is the minimum-sized rectangle that contains all points belonging to an object. The bounding box coordinates (x,y,w,h) indicate the (x,y) coordinates of the centre of the box, as well as the height (h) and width (w) of the box.*

While classification and localisation problems tend to only have one object per image, object detection problems may have multiple objects and multiple classes represented in a single image [50]. In an object detection problem, there is usually a fixed number of categories of interest. Given an input image, an object detector should determine whether there are instances of objects from the predefined categories. If one of those objects appears in a given input image, an object detector should draw a bounding box around the object and predict the category of the object [49].

4.2 Transfer learning

As mentioned in Section 3.2.3, the fully-connected layer can be adapted to accommodate cases where the localisation of an object within an image is required [41]. The softmax classifier layer can be replaced with a regression network and trained to predict bounding boxes; the other layers and parameters can be considered to be frozen. The classifier and regressor networks can then be run simultaneously to generate bounding box predictions [42]. The idea of using a pre-trained network for a new task is called transfer learning. It is particularly valuable in situations where there is relatively little data available for the new task [23]. Transfer learning generally involves taking an existing network architecture that has been trained on a large dataset such as ImageNet [51], MS COCO [52] or Pascal [53], and fine tuning it for a new task [54]. ImageNet, for example, contains 1000 object classes and 1.2 million images [23]. Neural network architectures that perform well on a given computer vision task often perform well on other computer vision tasks, as many of the features learnt during training — particularly lower-level features — are applicable across

multiple applications. Tuning an existing architecture using its pre-trained weights instead of random weight initialisations can reduce the time and computation requirements. If more training data is available, fewer layers may be frozen.

As object detection problems require training images that are annotated with both object categories and ground truth bounding boxes, the training images are more expensive to label than those used for pure classification problems. This means that there is typically less data available for object detection problems. Scarcity of labelled data means that there may be insufficient training images to train a large CNN [50]. It has been shown that transfer learning is particularly helpful in these situations, allowing a network that is pre-trained on a large dataset to be fine-tuned on a smaller dataset for a new application [50].

4.3 Evaluation criteria

In object detection tasks, there will be far more negative examples (background) than positive examples (objects). This imbalance means that it is not appropriate to use a simple accuracy measure [53]. This section discusses evaluation metrics that are relevant to object detection tasks.

4.3.1 Intersection over union (IoU)

Intersection over union (IoU) is a metric that can be used to describe how similar the predicted bounding box is to the ground truth bounding box. IoU is defined as the ratio between the intersection ($\cap$) and union ($\cup$) of the predicted bounding box (B_P) and the ground truth bounding box (B_{GT}), as shown in Eq. 4.1 [53, 55]:

$$\text{IoU} = \frac{\text{Area of } B_P \cap B_{GT}}{\text{Area of } B_P \cup B_{GT}} \tag{4.1}$$

Where $0 \leq \text{IoU} \leq 1$. The IoU is essentially a measure of the overlap of the predicted bounding box and the ground truth bounding box (Fig. 4.2). A perfect detection, where the predicted bounding box is identical to the ground truth bounding box, would have an IoU of one, while a predicted bounding box that does not overlap at all with the ground truth bounding box will have an IoU of zero. The IoU is used to determine whether a detection is a true positive (TP) or false positive (FP) [55]. A predicted bounding box is considered to be a TP if the IoU is greater than some user-defined threshold, usually at least 0.5 [53].

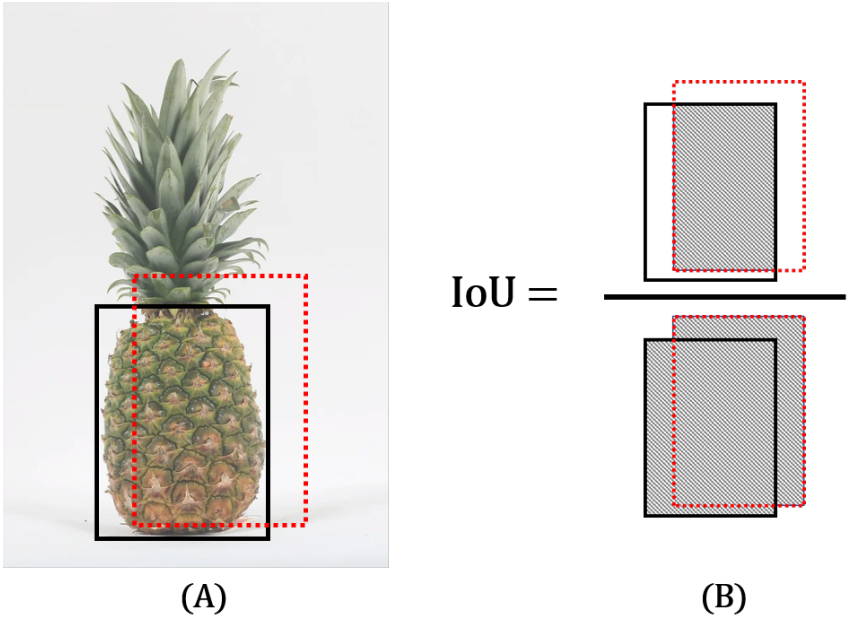

Figure 4.2: *Given an image (A) with a ground truth bounding box (solid black rectangle) and a predicted bounding box (dotted red rectangle), the IoU is determined as shown in (B), where the shaded area in the numerator represents the intersection (∩) of the predicted and ground truth bounding boxes, while the shaded area in the denominator indicates the union (∪) of the predicted and ground truth bounding boxes.*

4.3.2 Average precision (AP)

For a given IoU threshold, the TP and FP predictions can be used to determine the recall (Eq. 4.2) and precision (Eq. 4.3). Recall is the true positive rate — the ratio of true positive predictions to the number of actual (ground truth) positives — while precision is a measure of how many of the positive predictions are true positives.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\#GT} \tag{4.2}$$

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\#P} \tag{4.3}$$

Each predicted bounding box has an associated confidence level, which can be used to rank the output [53]. A precision/recall curve can then be computed from the ranked output [53]. The average precision (AP) summarises the shape of the precision/recall curve [53]. Multiple approaches for the calculation of average precision have been reported in literature [53, 52]. In the PASCAL VOC2007 challenge, the interpolated average precision [56] was used

as the evaluation metric. In this context, the AP summarises the shape of the precision/recall by taking the mean precision at a set of eleven equally spaced recall values [0, 0.1, ..., 1] [53].

In the later PASCAL VOC2010 challenge [53], the average precision (AP $\in [0, 1]$) of the detector is defined as the area under the precision/recall curve as calculated by numeric integration at a fixed IoU threshold of 0.5. This is calculated for each class and taking the mean of AP across all classes gives the mean Average Precision (mAP) [53]. This was an improvement on the interpolated AP, as it considered all recall levels instead of only 11 recall values.

The MS COCO Benchmark challenge [52] extended this by averaging the AP over a range of IoU values, from 0.50 to 0.95 at intervals of 0.05, denoted as $AP@[0.50 : 0.05 : 0.95]$. This rewards detectors with better localization. In this work, the MS COCO metric will be used to evaluate the performance of object detectors.

4.4 Detection frameworks

As the number of objects per image is not known beforehand, a standard CNN cannot be used for object detection, as it would have a fixed-length output. A simple way around this is to use the sliding window approach. With a sliding window detector, a window is moved across the input image to generate multiple smaller crops of the image. Each crop is then evaluated using a CNN to determine whether an object of interest is present in that area. However, there are several problems with this approach. Firstly, as objects might appear at any location in the image and have different aspect ratios or sizes, it would be necessary to perform an exhaustive search of all spatial locations, using a wide range of window sizes. This makes the method too computationally expensive. This shortcoming is resolved by the more modern approaches discussed in this section.

There are two major types of detection frameworks: region-based (two-stage) and unified (one-stage) frameworks [49]. Region-based frameworks are referred to as two-stage frameworks as they involve first generating category-independent region proposals before applying a classifier to determine the category labels of the proposed regions. On the other hand, one-stage detectors, such as You Only Look Once (YOLO) [57, 58, 59] and single shot detection (SSD) [60], feed forward in a single pass. As such, one-stage frameworks are often faster and more suited to online processing. However, they have been shown to have lower accuracy than two-stage frameworks such as Faster R-CNN [61]. In this work, fruit detection and size determination are required for fruit quality evaluation. However, as no online sorting is required, a higher accuracy is desirable. As such, this work will use a two-stage detector, namely Mask R-CNN. There were several different adaptions to get from the CNNs discussed in Chapter 3 to Mask R-CNN. In the subsections below, these adaptions will be explained.

4.4.1 Region-based CNN (R-CNN)

Region-based CNN (R-CNN) is so named because it combines region proposals with CNNs [50, 62]. While the sliding window approach to object detection required an exhaustive search of all spatial locations, region proposals identify regions of interest (RoI) where an object may be likely to occur. Hence, a more manageable number of candidate regions may be considered, reducing the number of locations at which the CNN must be evaluated [25].

A representation of an R-CNN is shown in Figure 4.3. The first stage of an R-CNN object detector is responsible for generating category-independent region proposals [50]. In the second stage of R-CNN, each RoI is passed through a CNN to obtain a fixed-length feature

vector [25, 50]. As the region proposals are of arbitrary size, it is necessary to warp each region to produce the fixed-sized inputs required by the CNN [62]. Finally, each region is classified using class-specific linear support vector machines (SVMs) [50].

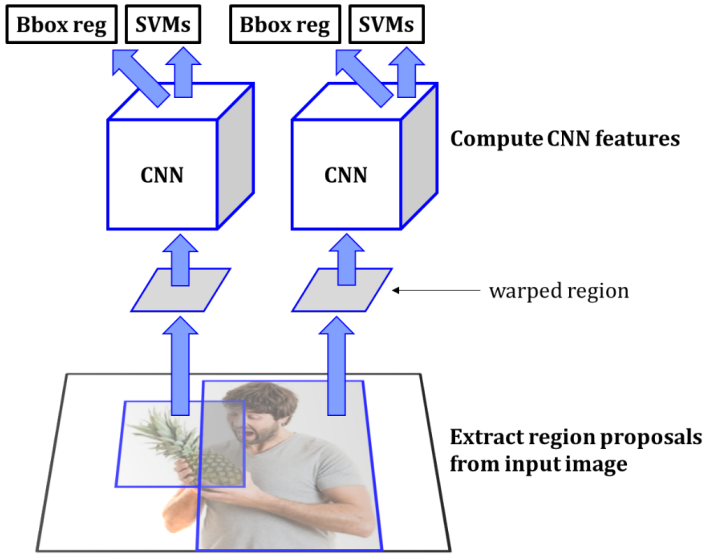

Figure 4.3: *R-CNNs use a region proposal network to extract around 2000 region proposals from an input image. The rectangular window around each region is warped to* 227×227 *before a large CNN is used to compute features for each region. Finally, each region is classified using class-specific linear SVMs and bounding box regression is used to update the bounding box coordinates. Adapted from [50].*

R-CNNs are agnostic to the region proposal method [50], allowing chosen any region proposal algorithm to be implemented. The region proposal algorithm that has been used extensively in practise is selective search [50, 63]. Selective search uses hierarchical grouping strategies to identify regions of interest that may be similar in terms of colour, texture, size and shape [63]. Using this method, only around 2000 region proposals are extracted [50], improving the computational cost of the problem with respect to the sliding window approach.

The RoI does not always correspond exactly with the ground truth bounding box. Hence, after applying the class-specific SVM, new bounding box coordinates are predicted using a linear regression model to reduce the localisation error [50, 62]. As R-CNN merely adjusts the bounding box coordinates rather than predicting object bounds itself, its accuracy is dependent on the performance of the region proposal algorithm chosen [22].

Although R-CNN is an improvement on the sliding window approach, training is expensive in terms of both space and time requirements [64].

4.4.2 Fast R-CNN

Fast R-CNN arose from extending R-CNN. Instead of processing each RoI separately, the whole image is run through the CNN, resulting in a convolutional feature map that corresponds to the whole image [64]. In Fast R-CNN (Fig. 4.4), the RoIs identified by selective search are projected onto the feature map. The RoIs on the feature map are then resized using the *RoI pooling layer* to give the fixed-size input expected by the subsequent fully connected (FC) layers [25].

The RoI pooling layer resizes the RoI on the feature map by dividing the RoI into a $H \times W$ grid of sub-windows, where $H \times W$ are hyperparameters. Max pooling is then applied to each of the sub-windows, giving a fixed-size feature map [25, 64]. As described in Section 3.2.2, pooling is applied independently to each channel of the feature map. Once pooled, each RoI is mapped to a feature vector by FC layers [64].

The RoI feature vector is then used as an input into a sequence of FC layers that branch into two output layers: one terminates in a softmax layer that is used to predict the class of the region, and the other is a bounding box regressor that gives offset values for the bounding box [64]. Fast R-CNN is faster than R-CNN because the convolution operation is only performed once per image, instead of on each of the proposed regions. Additionally, the use of a softmax classifier instead of training one-vs-rest linear SVMs [64]. These adaptions lead to increased speed and improved accuracy [25]. However, the region proposal computation was shown to be a bottleneck [22].

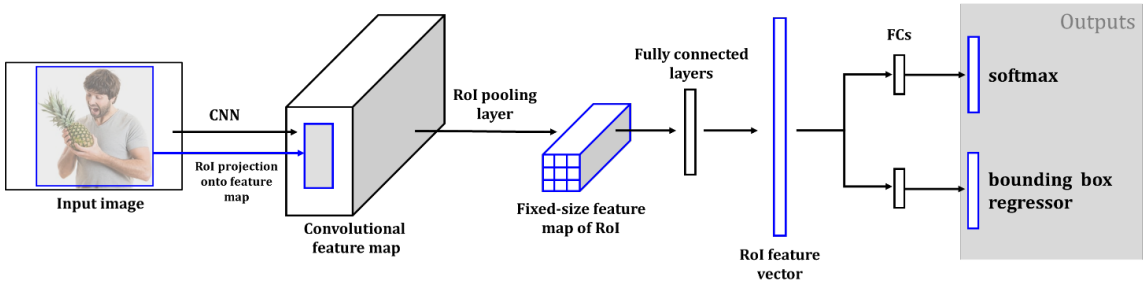

Figure 4.4: *Fast R-CNN processes the entire input image with a CNN to produce a feature map. Region proposals are projected onto the feature map and resized using an RoI pooling layer. Each RoI feature vector is passed through FC layers before branching into two output layers: one a softmax classifier and another that provides class-specific updated bounding box coordinates. Adapted from [64].*

4.4.3 Faster R-CNN

Both R-CNN and Fast R-CNN make use of the selective search algorithm for region proposals, which was identified as a bottleneck in Fast R-CNN [22, 50]. Faster R-CNN improves computation speeds relative to Fast R-CNN by eliminating the use of selective search, instead introducing a region proposal network (RPN) that shares convolutional features with the detection network [22]. The RPN that generates region proposals (i.e. proposed bounding boxes) is the first stage of the Faster R-CNN network [25]. The second stage is essentially Fast R-CNN, which extracts features using RoI pooling and performs classification and bounding box regression, as described in Section 4.4.2 [22, 25].

The aim of an RPN is to propose regions where an object is likely to be present in the input image. To propose bounding boxes, a small $n \times n$ network is moved across the convolutional feature map in a sliding-window fashion (Fig. 4.5) [22]. At each location of the sliding window, a maximum of k proposals – called anchor boxes – are generated [22]. These anchor boxes share the same centre point but have different scales and aspect ratios. If, for example, three scales and three aspect ratios are used, the number of anchors at each location will be $k = 9$ [22]. The anchors are used for reference when predicting the location of objects within an image. The set of k predefined anchor boxes are convoluted with each sliding window to produce fixed-length vectors that are then processed by two sibling layers that run in parallel. The box-classification layer (cls) is a binary softmax classifier indicating the class probabilities of each anchor box being an object or not an object. Hence, the cls layer has $2k$ output channels [65]. The box-regression layer (reg) has $4k$ output channels, as it encodes four bounding box coordinates for each of the k anchor boxes [22].

To eliminate redundancy when RPN proposals overlap over the same object, non-maximum suppression (NMS) based on cls scores can be used [22]. NMS discards proposed bounding boxes if they have a high level of overlap (determined by an IoU threshold) with another proposed bounding box that has a higher cls score [22]. This ensures that each object is only detected once. After NMS, the top-N ranked proposal regions are used for detection [22].

Faster R-CNN (Fig 4.6) combines a region proposal network and Fast R-CNN object detector that are trained simultaneously using four losses: the RPN has classification and bounding box regression losses, while the R-CNN detector has losses associated to the final classification of the object and the bounding box coordinates [22]. As convolutional features are shared by the RPN and the detection network, the region proposal step is nearly cost-free [22]. As the RPN is learnt, the quality of the region proposals is improved, resulting in improved performance in object detection accuracy, compared to previous methods [22].

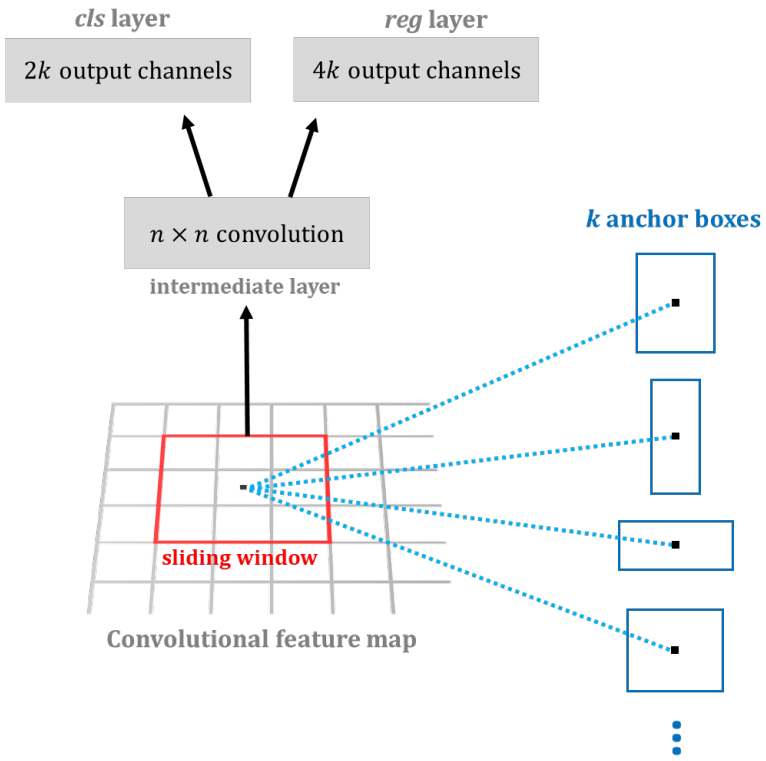

Figure 4.5: *Architecture of RPN in Faster R-CNN, using a sliding window of size $n = 3$. A set of k predefined anchor boxes are convoluted with each sliding window to produce fixed-length vectors that are then processed by the sibling cls and reg layers. The cls layer has $2k$ output channels, as it is a binary softmax classifier indicating the class probabilities of each anchor box being an object or not an object. The reg layer has $4k$ output channels, as it encodes four bounding box coordinates for each of the k anchor boxes [22].*

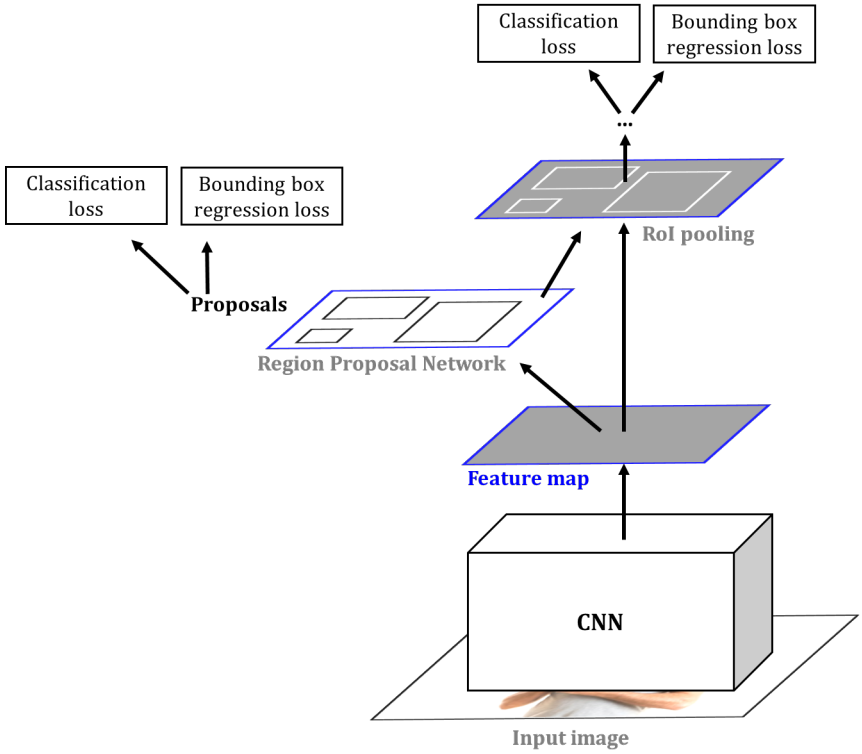

Figure 4.6: *Faster R-CNN combines a region proposal network and Fast R-CNN object detector that are trained simultaneously. The RPN proposes regions for the R-CNN to consider when detecting objects. Adapted from [22].*

4.4.4 Mask R-CNN

While all the object detection methods discussed up to this point identify located objects using a bounding box, Mask R-CNN takes it a step further by predicting a pixel-level object mask indicating the location of each object [25]. This is done by including a branch, in parallel to the one that performs bounding box regression and classification, which outputs a binary mask for each RoI. The mask branch is a small fully connected network that is applied to each RoI [25]. The resultant binary mask is a matrix with 1's indicating which pixels an object is present in. The task of locating each pixel belonging to an object, rather than simply localising using a bounding box, is referred to as instance segmentation [25] (Fig.4.7).

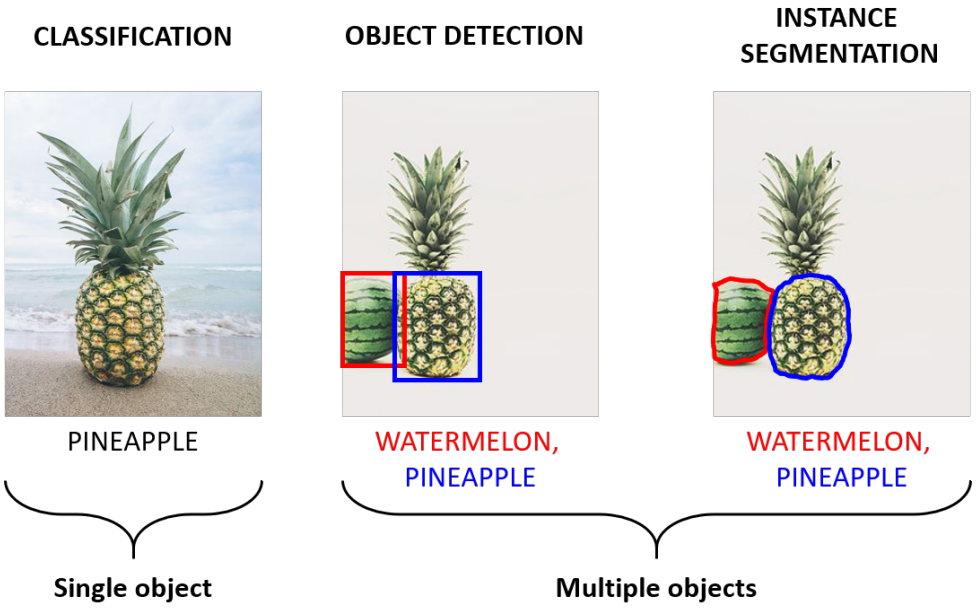

Figure 4.7: *While classification identifies a single main object in an image, object detection and instance segmentation both involve identifying all individual objects of interest in an image containing multiple objects. In object detection, objects are classified and localised using bounding boxes while in instance segmentation, all pixels belonging to an object are identified.*

Another significant difference between Faster R-CNN and Mask R-CNN is that RoIPool is replaced by RoIAlign (Fig. 4.8). Pixel-level segmentation requires alignment between the network input and outputs and RoIPool is not suited to this task as it involves quantisation, which leads to misalignment problems. RoIAlign, on the other hand, preserves exact spatial locations [25].

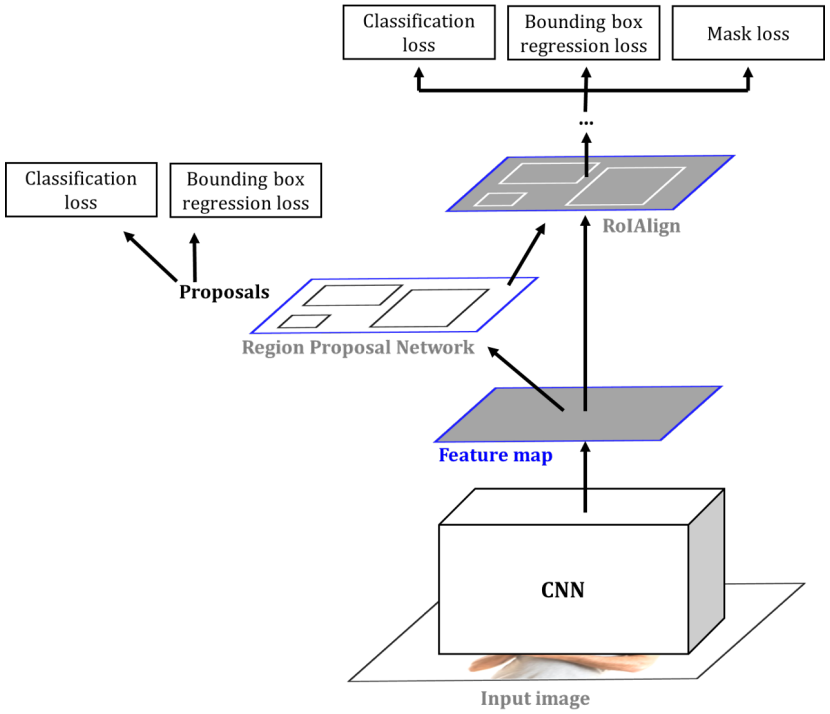

Figure 4.8: *Mask R-CNN extends Faster R-CNN by including a branch, in parallel to the branch that performs bounding box regression and classification, that outputs a binary mask. When training Mask R-CNN, a multi-task loss is defined using the classification loss, the bounding box regression loss, and the mask loss. Adapted from [25]*

During training of Mask R-CNN, a multi-task loss is defined on each RoI. The multi-task loss (Eq. 4.4) is the combination of the losses associated with the tasks of classification (L_{cls}), localisation (L_{bbox}) and instance segmentation (L_{bbox}) [25]:

$$L = L_{cls} + L_{bbox} + L_{mask} \tag{4.4}$$

If there are K classes, the classification loss, L_{cls}, is given by Eq. 4.5 [64]:

$$\begin{aligned} L_{cls} &= -log(p, u) \\ p &= (p_0, \ldots, p_K) \\ u &= 0, \ldots, K \end{aligned} \tag{4.5}$$

where $p = (p_0, \ldots, p_K)$ is the discrete probability distribution per RoI, as computed by a softmax (Eq. 3.11) over the $K + 1$ outputs of a fully connected layer. There are $K + 1$ outputs, as there is a catch-all background class, assigned as $u = 0$, in addition to the K classes. L_{cls} is then the negative log loss associated with the predicted class (p), given that the true class is u.

For the task of localisation, the bounding box regression layer outputs bounding-box regression offsets relative to the original RoI for each of the K object classes [64]:

$$L_{bbox}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^u - v_i) \tag{4.6}$$

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x|, & \text{otherwise} \end{cases} \tag{4.7}$$

where $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ is the predicted tuple for class u and $v = (v_x, v_y, v_w, v_h)$ is the actual bounding-box regression target [25]. $smooth_{L1}$ is a robust L1 loss that is reported to be less sensitive to outliers than the L2 loss used in R-CNN [25].

Lastly, the loss associated with the task of pixel-wise segmentation is defined as the average binary cross-entropy [25]:

$$L = -\frac{1}{m^2} \sum_{0 \leq i,j \leq m} [y_{ij} log \hat{y}_{ij}^k + (1 - y_{ij}) log(1 - \hat{y}_{ij}^k)] \tag{4.8}$$

where y_{ij} is the label of cell (i, j) in the true mask for a given $m \times m$ region and $\hat{y}_{ij}^k$ is the predicted label for the same cell (i, j) in the mask learned for the ground-truth class, k. That means that for an RoI associated with ground truth class k, L_{mask} is only defined on the k^{th} mask. As a mask is learnt for each class, there is no competition among classes [25].

Section 4.4 has introduced the adaptations made to CNNs that allow them to be used for object detection and instance segmentation applications. In the context of this work, instance segmentation with Mask R-CNN is used to locate all pineapples in an image. The size of each fruit will then be extracted from the predicted mask outputs of the Mask R-CNN.

Chapter 5

Methodology

This chapter covers details of the data collection and practical implementation of the computer vision models for pineapple detection and size determination. The main aim of this research is to determine pineapple fruit size from images. Considering the main aim of this research, there are two tasks that need to be accomplished. The first task involves the training and evaluation of a pineapple detector, while the second task involves determining the fruit size based on the object mask output of the detector. Given these two tasks, two separate image datasets are used in this work. This chapter first explains the data collection and annotation processes and then goes on to give the details of the image datasets used for the two main tasks. Details of the software implementation are then given. Thereafter, the model architecture for the instance segmentation task of identifying pineapples in images is described in detail, highlighting the differences between the considered models. Lastly, the approach to size determination is outlined and the statistical methods used for evaluation are explained.

5.1 Data collection and annotation

The image data used in this research comes from a pineapple juicing factory in the Eastern Cape of South Africa, where two conveyor belts carry the fresh pineapples into the factory for processing. Video footage was collected using two progressive scan CMOS cameras (Hikvision DS-2CD2145FWD-I(S)) located above the two conveyor belts (Camera A and Camera B) delivering pineapples to the factory, positioned directly overhead and parallel to the belts to prevent perspective distortion. VLC media player [66] was used to extract images from the video footage and the `opencv` library [67] was used to crop the images to a size of 970×605

pixels, such that only the two sections of the conveyor belt are visible in frame, as shown in Figure 5.1. This was done to reduce the effects of distortion, which was more prominent at the edges of the images. These cropped images were then manually labelled using the VGG Image Annotator (VIA) [68] – a tool used to draw polygon shapes around objects (Fig. 5.1). In all cases, the polygon tool was used at 2.5× level zoom. VIA [68] is a convenient tool as it runs in a web browser and does not require any installation or setup. The final output is a json file with the coordinates of each polygon shape. These polygon shapes are used to define the pixel-level binary masks required for training a Mask R-CNN: points located within the polygon are set to a value of one, indicating that an object is present at that location in the image, while pixels outside of the polygon are set to zero.

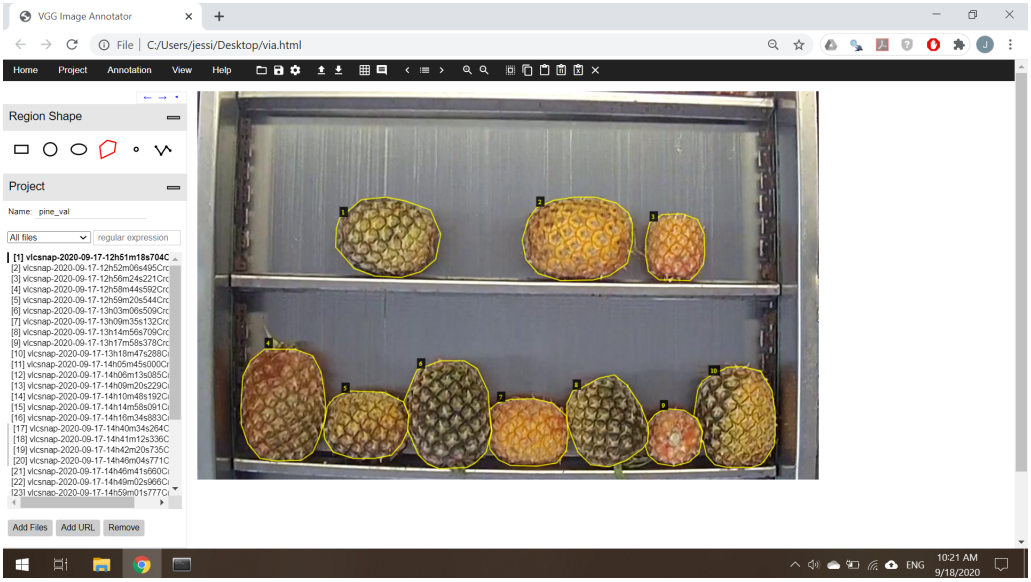

Figure 5.1: *Screenshot showing annotation using the VGG Image Annotator (VIA) tool [68]. Polygons were drawn around each pineapple visible in the image. Two sections of the conveyor belt are visible in each frame.*

5.2 Dataset for instance segmentation

For the purpose of training and evaluating a Mask R-CNN pineapple detector, a total of 160 images of size 970×605 pixels were extracted from the video footage as described in Section 5.1. These 160 images were randomly split into training, validation and test sets, ensuring that each set contained an equal number of images from Camera A and Camera B (Table 5.1). A 70/20/10 percent split was chosen for the training, validation and test sets. Due to resource limitations, only one validation set was used, rather than cross-validation. All images in the training set contained at least one fruit, and a total of 2138 pineapples were labelled. The number of pineapples per image varied between 2 and 30, with the number of pineapples per image being fairly similar throughout the three subsets (Fig. 5.2). By chance, the training set contains more complex images than the validation and test sets, with more pineapples per image in the training set than in the validation and test sets. The training set has a median of 14 pineapples per image, while the validation and test sets both have a median value of 11 (Fig. 5.2).

Subset	No. of images	%	# images Cam A	# images Cam B	Ave. # of pineapples per image	# of labelled pineapples
Train	112	70	56	56	14.2	1587
Validation	32	20	16	16	11.5	368
Test	16	10	8	8	11.4	183
Total	160	100	80	80	13.4	2138

Table 5.1: *Number of images in the train, validation and test sets used in the training and evaluation of various Mask R-CNNs for pineapple instance segmentation.*

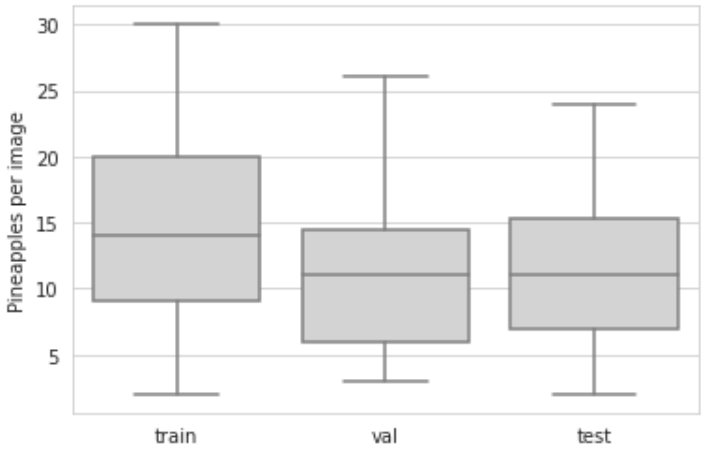

Figure 5.2: *Boxplot showing the number of pineapples per image for the training, validation and test sets. The median values are comparable across all subsets.*

5.3 Dataset for fruit size determination

Having detected the pineapples in an image using Mask R-CNN, the fruit size was found based on the object mask output of the detector; details of this process are given in Section 5.6.1. As the main aim of this research is to determine pineapple fruit size from images, it is necessary to assess whether the fruit sizes obtained from images are sufficiently similar to the fruit sizes that would be obtained by manual measurement of each pineapple. Hence, for evaluation of the size determination approach, images were obtained of 120 pineapples that had previously been manually measured.

As there are two conveyor belts delivering pineapples to the factory, 60 pineapples were placed on Conveyor A (Camera A) and 60 pineapples were placed on Conveyor B (Camera B). Video footage was captured as they passed under the cameras and images were obtained from video footage as described in Section 5.1. These images had not previously been seen by the model as they had not been used in the training or validation of the Mask R-CNN. The 120 pineapples had been pre-measured manually, using a calliper system (Fig. 5.3). The length and width of each fruit were recorded, as well as the weight, which was measured using a balance (CAS PR PLUS, accurate to 0.002 kg). These measured values are summarised in Figure 5.4, where the histogram for each metric is given, together with the Gaussian kernel density estimates (KDE), to better visualise the shape of each distribution. The methods used to evaluate the similarity of the predicted sizes and the actual measurements will be discussed in detail in Section 5.6.2.

Figure 5.3: *Calliper system used for manual measurement of pineapples for size determination evaluation.*

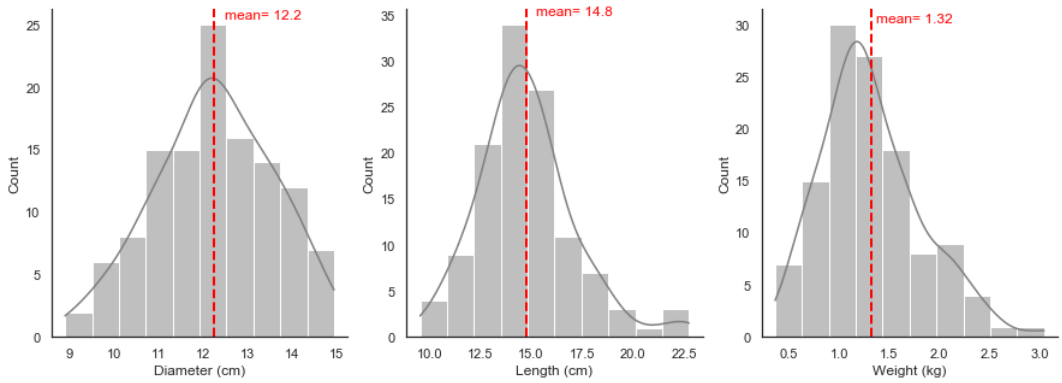

Figure 5.4: *Histogram with KDE (solid line) overlaid, showing the distribution of the diameter (left), length (centre) and weight (right) of the 120 hand-measured pineapples. The red dotted line shows the mean value in each case.*

5.4 Software implementation

Training of Mask R-CNN models was done using a free GPU accelerator on Google Colaboratory, commonly referred to as "Google Colab" [69]. In all cases, the Matterport Mask R-CNN implementation is used [70]. This implementation of Mask R-CNN makes use of Python 3, the Keras [71] application program interface (API), and the TensorFlow [72] library. The model is based on Feature Pyramid Network (FPN) [73] and a ResNet [74] CNN backbone, and generates both bounding boxes and segmentation masks for each instance of an object in each image [70]. The FPN component allows for detecting objects at different scales [73].

5.5 Instance segmentation

Several instance segmentation models were considered in this work. The pineapple detection performance was assessed with respect to (i) transfer learning using pretrained COCO and ImageNet weights, (ii) the choice of CNN backbone, and (iii) data augmentation techniques.

5.5.1 Transfer learning

In computer vision tasks, it has become common practise to pre-train a CNN on a large dataset and then transfer the learned features to a new task that has a smaller number of labelled examples [23]. In many cases, starting weights are readily available for different network architectures, particularly those pre-trained on the ImageNet [51] and MS COCO [52] databases, which are commonly used in image recognition tasks. These pre-trained weights are used to initialise the parameters of the network before training on a new task. While low level features are likely to be similar across different datasets, higher level features learnt by deeper layers in the CNN network are more specific to the task at hand [23, 75]. Therefore, the performance of an object detector is affected by the similarity of the new target classes to the base classes used for pre-training [75].

While the ImageNet dataset contains an unprecedented number of images, MS COCO (Microsoft: Common Objects in Context) arose in response to the criticism that images in the ImageNet dataset tend to contain large and well-centred objects, making the ImageNet dataset unrepresentative of real-world scenarios [49, 52]. Although the MS COCO dataset is smaller than ImageNet, it contains more complex scenes, with varied viewpoints of common objects in their natural environments [49, 52]. A comparison of ImageNet and MS COCO is given in Table 5.2 [49].

In this section of work, the performance of two models trained on the same pineapple dataset (Section 5.2) with different starting weights is investigated. Model 1 makes use of ImageNet features (ImageNet_NoAug_Res101), while Model 2 makes use of MS COCO features (COCO_NoAug_Res101).

Name of dataset	Total images	Categories	Objects per image	Image size	Year started	Ref
Imagenet	∼ 14 m	21 841	1.5	500 × 400	2009	[51]
MS COCO	∼ 328 k	91	7.3	640 × 480	2014	[52]

Table 5.2: *Comparison of ImageNet and MS COCO – popular databases for object recognition.*

5.5.2 CNN backbone

With the Matterport Mask R-CNN implementation [70] used in this work, it is possible to choose between ResNet-50 and ResNet-101 CNN backbones. A ResNet is a "residual network", characterised by residual blocks containing shortcut connections that perform identity mapping (Fig. 5.5) [74]. Prior to the introduction of ResNets, very deep networks suffered from a problem of vanishing gradients, which resulted in deeper networks having higher training errors than their shallower counterparts. However, the introduction of residual blocks ensures that information from earlier layers is retained [74]. ResNet-50 and ResNet-101 have similar structure, with both employing the concept of residual blocks. However, ResNet-50 has 50 layers, while ResNet-101 has 101 layers.

To investigate the effect of employing different CNN backbones, Model 2, which has a ResNet-101 CNN backbone (COCO_NoAug_Res101), was compared to Model 3, which has a ResNet-50 backbone (COCO_NoAug_Res50). Both models were trained on the same pineapple dataset (Section 5.2) and used the same starting weights.

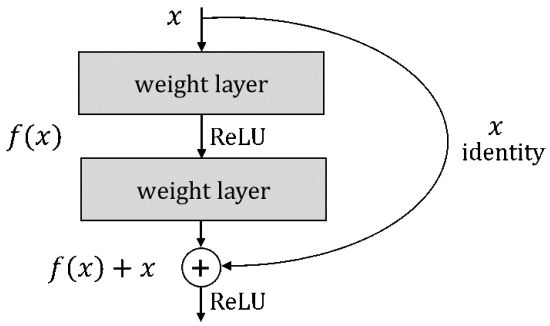

Figure 5.5: *A residual block.*

5.5.3 Data augmentation

In cases where not much labelled training data is available, a common method to avoid overfitting on the available data is to artificially enlarge the dataset, and the variability thereof, using label-preserving transformations [39]. This process is referred to as data augmentation [23] and can be performed using a Python library such as `imgaug` [76]. Frequently used augmentation techniques include horizontal mirroring, random cropping, rescaling, and colour shifting [23]. Distortions to the colour channel help the model become more resistant to changes in illumination. While it is possible to use data augmentation to expand the dataset with copies of the augmented versions, it is preferable to randomly augment the data with each training epoch [23]. This work investigates the effect of gaussian blurring and noise, horizontal flipping and colour shifting augmentation techniques (Fig. 5.6).

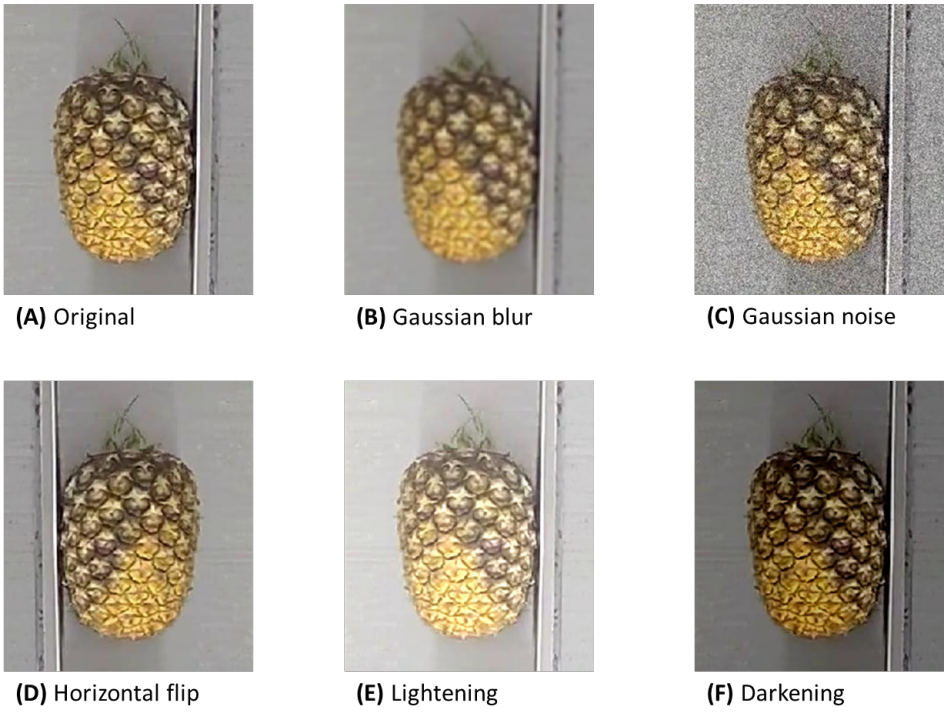

Figure 5.6: *Demonstration of augmentations applied using `imgaug` library.*

5.6 Size determination

In this section, the two approaches to size determination considered in this work are described. Thereafter, the methods of evaluating these size determination approaches are outlined.

5.6.1 Approaches to size determination

As discussed in Chapter 2, there are several different ways to describe the size of a fruit. Common metrics include weight, projected area, and dimensions such as length and diameter [2]. Historically, size measurements recorded at the pineapple factory have included diameter, length and weight. In this work, two approaches to size determination are employed.

The first approach involved extracting the diameter and length of pineapples from the predicted masks. In this approach, the Mask R-CNN trained to identify pineapples was used to predict masks for pineapples (Fig. 5.7 (A)) in the previously-unseen dataset of pineapple images, described in Section 5.3, to post-validate the size determination approach. The model outputs a binary mask for each detected object (Fig. 5.7 (B)). The OpenCV [67] library's `minAreaRect()` function was used to find the diameter and length of each fruit by fitting a minimum area rotated rectangle to the mask (Fig. 5.7 (C)). However, this function could not be applied directly to the binary mask as a 2D array, so the `findContours()` function from OpenCV was first used to extract the coordinates of the points describing the polygon outline of the binary mask.

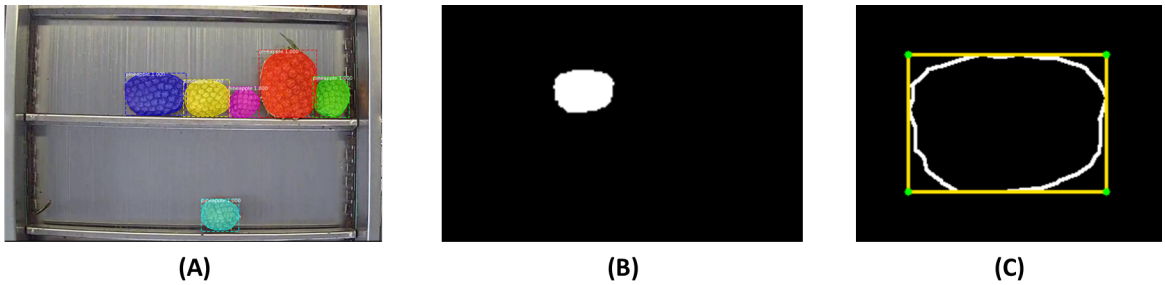

Figure 5.7: *(A) RGB image showing the different masks predicted by a Mask R-CNN trained on the pineapple dataset. (B) Each detection is outputted as its own binary mask. (C) The points describing the polygon outline of the mask were found, and the length and diameter measurements of the fruit were then extracted from the dimensions of the minimum rotated rectangle (shown in yellow) enclosing those points.*

The second approach to size determination involved finding the projected area of each fruit. As the mask output of the Mask R-CNN model contains a binary mask for each object instance, this can be achieved by summing pixels over each mask layer. To evaluate this method, the pixel area of the detected mask is compared to the pixel area of the ground truth mask. As the ground truth mask is labelled by a human, this method gives an indication of the best human performance in terms of determining fruit size from images.

5.6.2 Evaluation of size determination approach

The previously unseen dataset used for evaluation of the size determination approaches contains images of 120 pineapples that had been manually measured prior to image acquisition. The distributions of measurements obtained from detected masks, as outlined in Section 5.6.1, was compared to the distributions of the manually measured values in order to establish the efficacy of the size determination approach.

Histograms were plotted for each scaled measurement (diameter, length, projected area), using a bin number of 10. The number of bins was chosen by considering the square root method [77], in which the number of bins (k) is given by the square root of the number of observations in the set ($\sqrt{n}$), as well as the Rice Rule, which states that $k = 2\sqrt[3]{n}$ [78]. As there are 120 observations in the set, the number of bins determined by the square root was 11, while the number of bins determined by the Rice Rule was 10.

There are several statistical tests that can be used to show whether two distributions are the same. These include the two-sample Z-test, and the Kolmogorov–Smirnov test (KS test). The two-sample Z-test is a parametric method used to test the means of two distributions. A parametric test is a form of hypothesis testing in which assumptions are made about the underlying distribution of observed data [79]. In the case of a Z-test, it is assumed that the two samples are normally distributed [80]. However, when the sample size is sufficiently large ($n > 30$), the sample is expected to be approximately normally distributed because of the central limit theorem [81].

The two-sample Z-test is used to determine whether two samples come from the same population. If two samples are from the same population, it is expected that their sample means will be the same ($\mu_1 = \mu_2$). When testing for equivalence of means, the null hypothesis is that the difference between population means is zero [82], while the alternative hypothesis is that the population means are not equal (Eq. 5.1 [80]).

$$\begin{aligned} H_0 &: \mu_1 - \mu_2 = 0 \\ H_1 &: \mu_1 - \mu_2 \neq 0 \end{aligned} \tag{5.1}$$

The Z-statistic evaluates the difference between the two samples and is calculated according to Equation 5.2 [80, 83].

$$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \tag{5.2}$$

where $\bar{X}_1$ and $\bar{X}_2$ are the means of samples one and two; σ_1 and σ_2 are the population variances for samples one and two; and n_1 and n_2 are the number of observations in samples one and two, respectively. When population variances are unknown, the sample variances s_1^2 and s_2^2 can be used in Equation 5.2 instead, provided that both n_1 and n_2 are greater than 40 [80]. The p-value [84] is the probability of observing a difference in the means between the two distributions at least as extreme as the observed outcome, given that the null hypothesis is true.

The Kolmogorov–Smirnov test (KS test), on the other hand, is a non-parametric approach and can therefore be used to test whether any two arbitrary distributions are the same, without making any assumptions about their underlying shape of the distribution [85, 86]. The KS test tests the equality of the distributions of two samples and is based on comparing two empirical cumulative distribution functions (ECDFs) [87, 88]. For each value of the measured variable, an ECDF returns the fraction of observations of the measured variable that are less than or equal to that value [82, 86]. That is, it returns the probability that the measured variable is less than or equal to a given value.

The KS test compares two distributions by identifying the largest absolute difference between their ECDFs [85] (Fig. 5.8). Mathematically, for two ECDFs, F^1 and F^2, the KS test computes the KS statistic as shown in Equation 5.3 [87, 88, 89]:

$$KS = \max|F^1_{n_1}(x) - F^2_{n_2}(x)| \tag{5.3}$$

As ECDFs give probabilities, their values lie between 0 and 1 and two distributions that do not overlap at all will have a KS value of 1. Hence, large values of KS, close to 1, suggest that the two distributions are not equal, while values closer to zero indicate that the two distributions have similar shapes [86].

In this work, having considered Mask R-CNN pineapple detectors with different starting weights, CNN backbones, and data augmentation strategies, the best-performing detector was selected, and its mask outputs were used to determine the size of 120 previously-unseen pineapples from images. The two-sample Z-test and the Kolmogorov–Smirnov test (KS test) were then used to establish whether the fruit sizes obtained from images are sufficiently similar to the fruit sizes obtained by manual measurement of each pineapple.

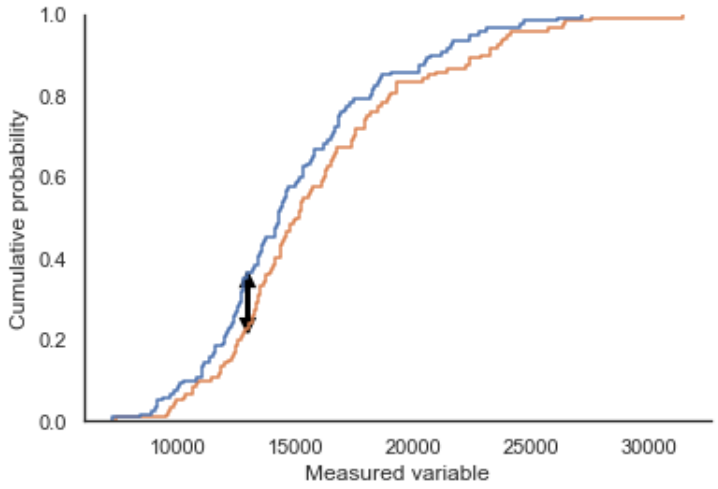

Figure 5.8: *Comparison of two ECDFs (blue and orange lines). The two-sample KS statistic, showing the maximum discrepancy between the two distributions, is indicated with a black arrow.*

Chapter 6

Results

This chapter discusses the performance of the detection models, as well as the efficacy of the size determination approach. In Section 6.1, the evaluation metrics described in Section 4.3 are used to assess various Mask R-CNN models to identify the best-performing pineapple detector to be used in this research. Thereafter, in Section 6.2, statistical methods are used to determine whether the fruit sizes obtained from images are representative of their actual measurements.

6.1 Object detection

The aim of the first task in this research was to train an instance segmentation model to effectively localise pineapples within images. To this end, several Mask R-CNN models were considered. There are many parameters that can affect the performance of a detector, including (i) choice of the starting weights, (ii) choice of CNN backbone, and (iii) whether data augmentation is used.

6.1.1 Transfer learning: choice of starting weights

Pineapple detection performance was assessed using two transfer learning procedures. The first involved initialising the Mask R-CNN network weights with ImageNet features (Model 1: ImageNet_NoAug_Res101), while the second initialised network weights with MS COCO features (Model 2: COCO_NoAug_Res101). The training and validation losses for these two choices of starting weights are shown in Figure 6.1, where it is evident that the losses asso-

ciated with Model 2, initialised with COCO pretrained weights, are considerably lower than those of Model 1, which was initialised with ImageNet weights. In Figure 6.1, it is evident that the validation loss is lower than the training loss. This may arise because the validation set contains easier examples than the training set. In this case, the validation set contains only 32 images, with fewer pineapples per image than the training set, as seen in Figure 5.2. This can happen by chance when the validation set is small.

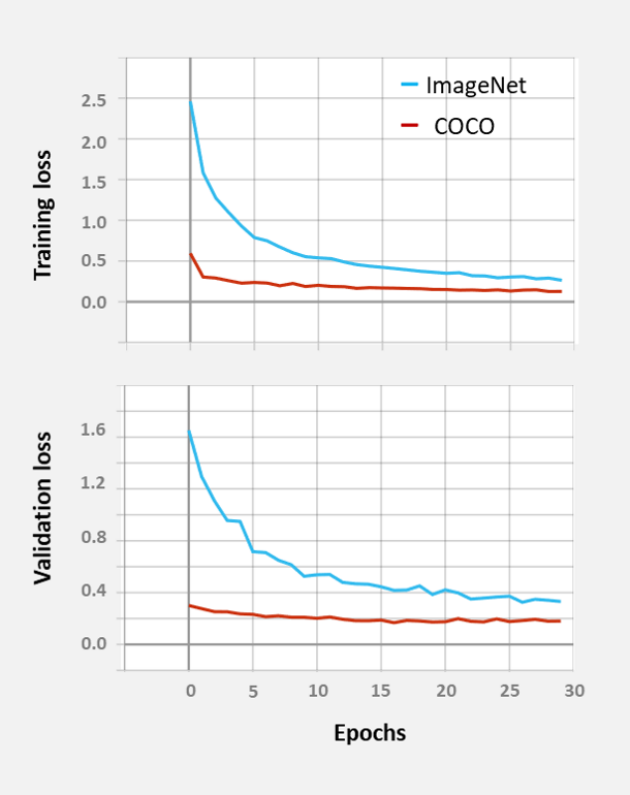

Figure 6.1: *Training and validation set losses for Mask R-CNNs initialised with ImageNet weights (blue) and COCO weights (red). Training losses are shown in the top panel, while validation losses are shown in the bottom panel.*

The ImageNet dataset contains far more images than the MS COCO dataset (Table 5.2) and yet it was outperformed by COCO in this context. This may be due, in part, to the fact that the ImageNet dataset only has an average of 1.5 objects per image, while the COCO dataset has an average of 7.3 objects per image, which is more comparable to the average number of objects per image in the pineapple dataset. The pineapple dataset used for instance segmentation had an average of 13.4 pineapples per image (Table 5.1).

The AP performance of the two models on the validation set is summarised in Table 6.1. Both models performed well in terms of AP@0.5. However, the COCO-initialised model, COCO_NoAug_Res101, achieved an AP@[0.50:0.05:0.95] value of 0.892, compared the to the ImageNet-initialised model, ImageNet_NoAug_Res101, which achieved an AP@[0.50:0.05:0.95] value of 0.860. As the MS COCO features were better suited to the task of identifying pineapples, these were chosen as starting weights for all subsequent networks.

Model #	Model name	Starting weights	# epochs	Validation AP	
				IoU= 0.5	IoU= 0.50:0.05:0.95
1	ImageNet_NoAug_Res101	ImageNet	30	1.00	0.860
2	**COCO_NoAug_Res101**	**COCO**	30	0.998	**0.892**

Table 6.1: *Validation average precision (AP) summary of two Mask R-CNNs for pineapple detection trained using different transfer learning procedures. Model 1 was initialised with ImageNet starting weights, while Model 2 was initialised with MS COCO starting weights. Both models have a ResNet101 CNN backbone and did not employ data augmentation.*

6.1.2 CNN backbone

Mask R-CNN pineapple detectors with different variations of the ResNet CNN backbone were assessed. Model 2 (COCO_NoAug_Res101), with its ResNet101 backbone was compared to Model 3 (COCO_NoAug_Res50), which made use of a smaller ResNet50 backbone. The only difference between these two models is their CNN backbone; all other parameters were kept the same during training and both models were initialised using MS COCO features. The training and validation losses for the two choices of CNN backbone were very similar, as can be seen in Figure 6.2.

A comparison of the AP performance of Model 2 (COCO_NoAug_Res101) and Model 3 (COCO_NoAug_Res50) (Table 6.2) shows that both models performed well in terms of AP@0.5. Interestingly, Model 3 (COCO_NoAug_Res50) performed almost as well as its larger counterpart in terms of AP@[0.50:0.05:0.95] (Table 6.2), achieving a value of 0.884, compared to Model 2 (COCO_NoAug_Res101), which achieved an AP@[0.50:0.05:0.95] value of 0.892. As Model 3, with its smaller CNN backbone, had comparable performance to that of the larger network, it was chosen for subsequent steps in this work.

Model #	Model name	CNN backbone	# epochs	Validation AP	
				IoU= 0.5	IoU= 0.50:0.05:0.95
2	COCO_NoAug_Res101	ResNet101	30	0.998	0.892
3	**COCO_NoAug_Res50**	**ResNet50**	30	1.000	**0.884**

Table 6.2: *Validation average precision (AP) summary of two Mask R-CNNs for pineapple detection with different CNN backbones. Model 2 has a ResNet101 backbone, while Model 3 makes use of ResNet50 architecture. Both models were initialised with MS COCO starting weights and did not employ data augmentation.*

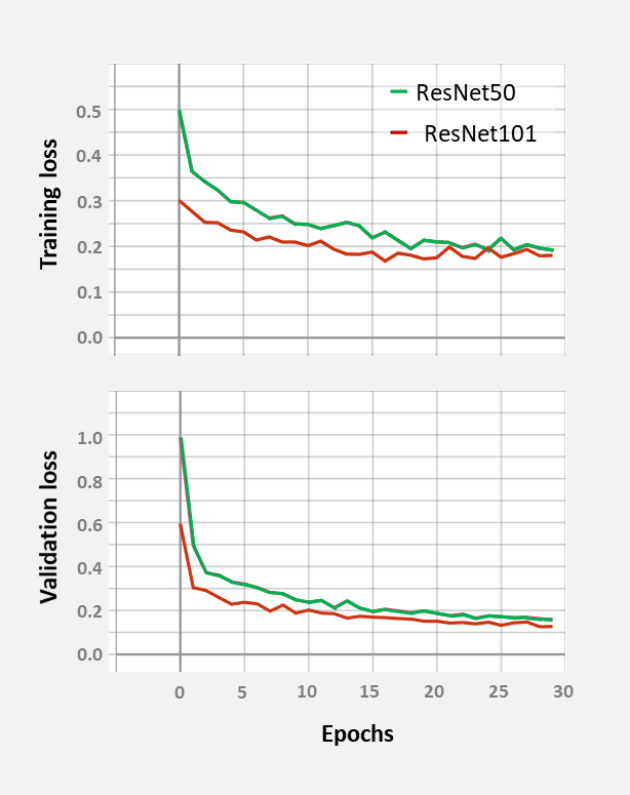

Figure 6.2: *Training and validation set losses for Mask R-CNNs with ResNet50 CNN backbone (green) and ResNet101 CNN backbone (red). Training losses are shown in the top panel, while validation losses are shown in the bottom panel.*

6.1.3 Data augmentation

Data augmentation was applied with the pineapple training set using horizontal flip (Model 4: COCO_Fliplr_Res50), Gaussian noise and Gaussian blur (Model 5: COCO_GaussNB_Res50), lightening and darkening (Model 6: COCO_Colour_Res50) and a combination of the above three approaches (Model 7: COCO_All_Res50). These were all compared with respect to Model 3 (COCO_NoAug_Res50), which did not make use of data augmentation during training. All models were initialised using MS COCO features and had a ResNet50 CNN backbone. The training and validation losses for models trained using the different data augmentation strategies outlined above are shown in Figure 6.3. The use of data augmentation resulted in a decrease in both training and validation set losses, although all augmentation strategies appear to have similar performance (Fig. 6.3). A summary of the average training and validation set losses (Table 6.3) shows that Model 4 (COCO_Fliplr_Res50) has the lowest training loss and validation loss, on average. Table 6.4 confirms that Model 4 has the best performance, with an AP@[0.50:0.05:0.95] of 0.914.

Model #	Model name	Augmentation	Average training loss	Average validation loss
3	COCO_NoAug_Res50	None	0.261	0.250
4	COCO_Fliplr_Res50	Horizontal flip	**0.158**	**0.146**
5	COCO_GaussNB_Res50	Gaussian noise and blur	0.161	0.151
6	COCO_Colour_Res50	Lightening & darkening	0.174	0.164
7	COCO_All_Res50	All of the above	0.179	0.155

Table 6.3: *Summary of training and validation set losses for Mask R-CNNs using different data augmentation strategies. All models have a ResNet50 CNN backbone and were initialised with MS COCO starting weights. The average training and validation losses are calculated across all 30 epochs. For each model, the minimum validation loss is reported, together with the epoch in which the minimum value was achieved.*

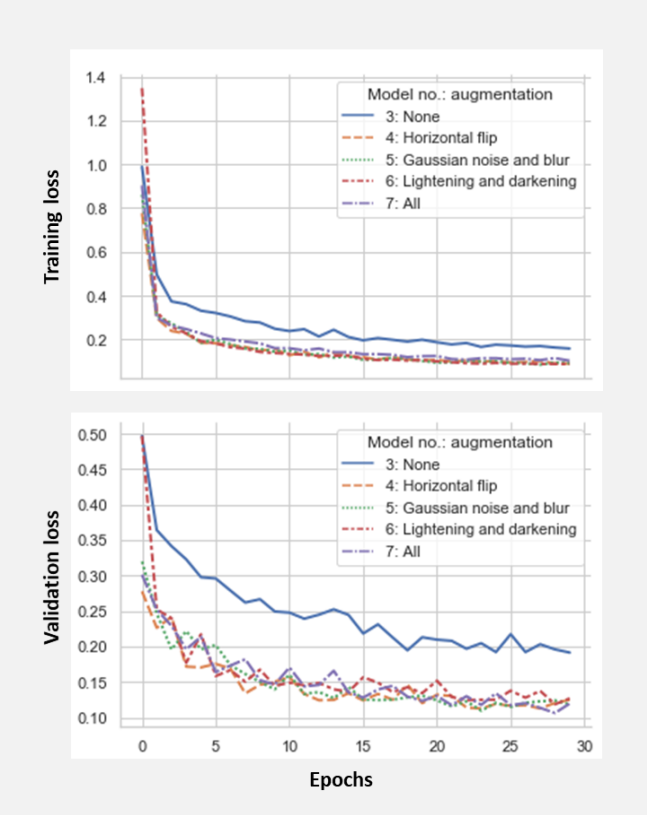

Figure 6.3: *Training and validation set losses for Mask R-CNNs employing different data augmentation strategies. Training losses are shown in the top panel, while validation losses are shown in the bottom panel. Models 4–7 were trained using data augmentation and had lower training and validation losses than Model 3, which did not make use of data augmentation.*

Model #	Model name	Augmentation	Min. Val. Loss	# epochs	Validation AP	
					IoU= 0.50	IoU= 0.50:0.05:0.95
3	COCO_NoAug_Res50	ResNet50	0.192	30	1.000	0.884
4	**COCO_Fliplr_Res50**	Horizontal flip	0.113	28	1.000	**0.914**
5	COCO_GaussNB_Res50	Gaussian noise and blur	0.110	24	1.000	0.905
6	COCO_Colour_Res50	Lightening & darkening	0.119	29	1.000	0.898
7	COCO_All_Res50	All of the above	0.107	29	1.000	0.898

Table 6.4: *Validation average precision (AP) summary of Mask R-CNNs pineapple detectors employing different data augmentation strategies during training. All models have a ResNet50 CNN backbone and were initialised with MS COCO starting weights. For each model, the weights associated with the epoch with lowest validation loss were used to determine the AP.*

6.1.4 Test set performance and error cases

Model 3 (COCO_NoAug_Res50) was considered the best model that did not make use of data augmentation, while Model 4 (COCO_Fliplr_Res50) was identified as the overall best pineapple detector considered in this work, based on validation AP values. As such, the performance of these two models on the withheld test set was evaluated. The pineapple detector Model 3 (COCO_NoAug_Res50) achieved a test AP@0.50 of 0.997 and a test AP@[0.50:0.05:0.95] of 0.874. However, as expected, Model 4 (COCO_Fliplr_Res50) improved upon this, achieving a test AP@0.50 of 1.000 and a test AP@[0.50:0.05:0.95] of 0.901.

For the best models with augmentation (Model 4) and without augmentation (Model 3), a search was performed to identify images in the test set where the number of detected masks was not equal to the number of ground truth masks. Fig. 6.4 (A), captured by Camera A, highlights an example where a pineapple was successfully detected by Model 4 but had not been detected by Model 3. The pineapple in question was partially occluded by another pineapple that was lying on top of it.

For both Model 3 and Model 4, one image was identified that did not have an equal number of detected and ground truth masks. In this image, there was one less ground truth mask than detected masks. This means that the models detected a pineapple that had not been annotated. This extra detection is shown in Figure 6.4 (B), captured by Camera B. In this case, the detected object is in fact a pineapple that had not been labelled as it was primarily out of frame. Comparing images taken with Camera A and Camera B (Fig. 6.4), we see that the heights of the two cameras are slightly different. In images taken with Camera A, only the contents of the 2 conveyor sections are in frame (Fig. 6.4 (A)). However, in images taken with Camera B, the adjacent sections are slightly visible at the top and bottom of the image, and some background is visible to the left and right (Fig. 6.4 (B)).

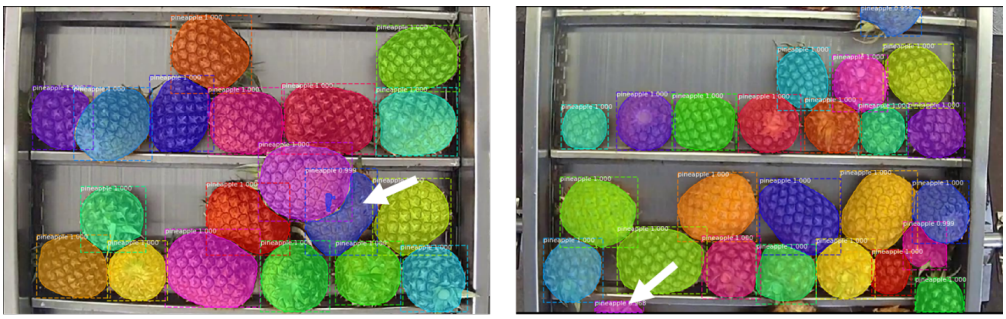

Figure 6.4: *Test set images showing the masks detected by Model 4 (COCO_Fliplr_Res50). (A) shows an image from Camera A. The pineapple indicated by the white arrow was successfully detected by Model 4 but had not been detected by Model 3. (B) shows an image from Camera B. One pineapple, indicated by the white arrow, was detected by the network but had not been labelled as it did not appear within the two conveyor sections centred in frame.*

6.2 Size determination

As Model 4 (COCO_Fliplr_Res50) was determined to be the best pineapple detector considered in this work, it was the model chosen for determining the size of pineapples from images. As such, Model 4 (COCO_Fliplr_Res50) was used to predict masks for pineapples in the previously-unseen dataset of pineapple images (described in Section 5.3), to post-validate the size determination approach. The process described in Section 5.6.1 was used to find the length and diameter of each fruit. The dimensions obtained by manual measurement using callipers were compared to the dimensions extracted from the predicted object masks by plotting histograms of the scaled data (Fig. 6.5). Visually, the predicted measurements seem to resemble the actual measurements.

For both the fruit diameter and fruit length, two-sample Z-tests were used to indicate whether the mean of the dimensions obtained by manual measurement is equal to the dimensions extracted from the predicted object masks. Z-tests for diameter measurements (Z=0.000, p=1.000, two-tailed) and length measurements (Z=0.000, p=1.000, two-tailed). The two-sample Z-tests for both the diameter and length had very high p-values, indicating that there is not enough evidence to reject the null hypothesis that the two population means are equal.

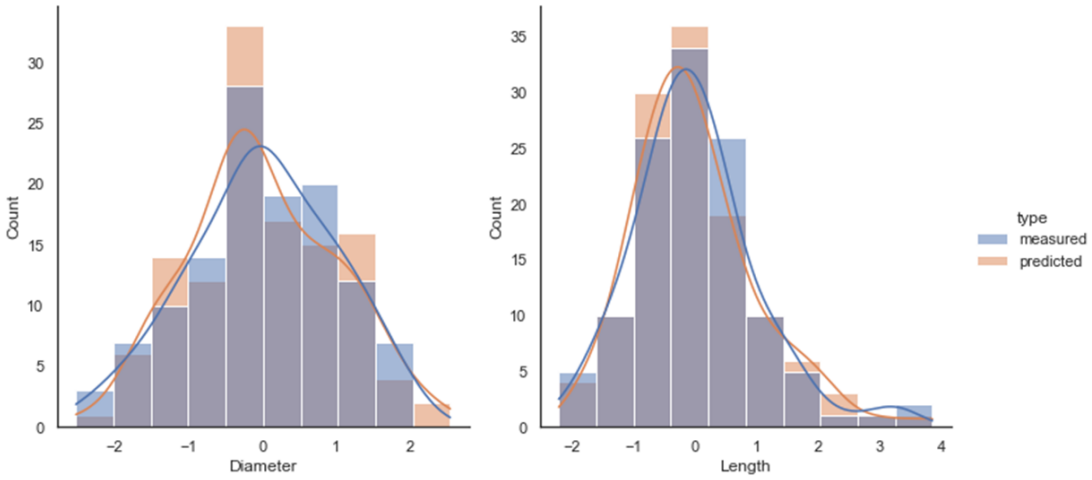

Figure 6.5: *Scaled distributions for fruit diameter (left) and length (right). The dimensions obtained by manual measurement using callipers are shown in blue bars, while the dimensions obtained from the predicted object masks are shown in orange.*

To further investigate whether the distribution of predicted fruit dimensions is the same as the distribution of actual, hand-measured dimensions, we consider the empirical cumulative distribution functions (ECDFs) shown in Figure 6.6.

The ECDF of the scaled predicted fruit diameter has the same general shape and position as the ECDF of the scaled manually measured diameter. A two-sample KS test showed that the distributions of the measured and predicted diameters were found to have a KS value of 0.083, and a p-value of 0.801. As the KS test had a high p-value ($p \gg 0.10$), there is not enough evidence to reject the null hypothesis that the two distributions are equal. According to this test, the difference between the two distributions is not significant enough to say that they are explicitly different.

A two-sample KS test showed that the distributions of the measured and predicted lengths were found to have a KS value of 0.058, and a p-value of 0.987. The high p-value indicates that there is not enough evidence to reject the null hypothesis of there is no difference between the two distributions.

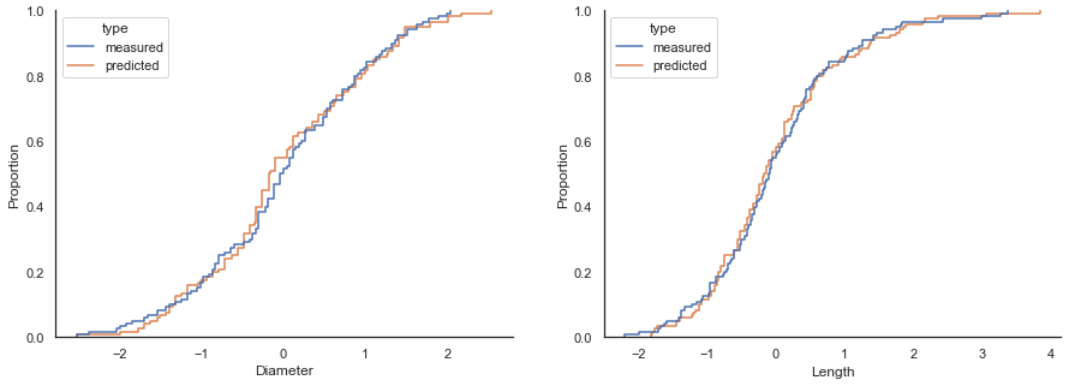

Figure 6.6: *Empirical cumulative distribution functions (ECDFs) for the scaled diameter (left) and scaled length (right). The dimensions obtained by manual measurement using callipers are shown by blue lines, while the dimensions obtained from the predicted object masks are shown in orange. Visually, the ECDFs for the predicted measurements seem similar to those of the actual measurements.*

Figure 6.7 shows some examples of images used for size determination where some pineapples are seen to be standing upright. In these cases, the longitudinal section of the pineapple is not visible, resulting in underestimation of length measurements.

Figure 6.7: *In some images used to evaluate the size determination approach, some pineapples (circled in white) are seen to be standing upright rather than lying flat. For these fruits, the algorithm is unable to extract accurate length measurements.*

The second approach to size determination involved finding the projected area of each pineapple instance. The projected areas extracted from the predicted object masks were compared to the projected areas of the hand-labelled ground truth masks. As both the predicted and ground truth masks had the same shape, the values were not scaled. The projected areas are given as a pixel area, based on a resized image of dimension 1024×1024 px. As the predicted masks are being compared to the ground truth masks, rather than the actual fruit measurements, differences in metrics should not arise due to the orientation problem highlighted in Figure 6.7. Hence, comparison of the projected areas is useful, as the ground truth area gives an indication of what the human performance would be on this task of determining fruit size from images.

The histograms in Fig. 6.8 (A) have very similar shapes but the predicted projected area is shifted to the left of the ground truth projected area. This phenomenon can also be observed from the ECDFs in Fig. 6.8 (B), where the shape of the predicted area ECDF closely resembles that of the ground truth area ECDF but is somewhat offset. While the shapes of the KDEs and ECDFs are very similar, the predicted projected areas are shifted to the left of the ground truth areas, indicating that the detector tends to under-report the projected area. The two-sample Z-test for projected area (Z=-1.491, p=0.136, two-tailed) had a fairly high p-value, indicating that there is not enough evidence to reject the null hypothesis that the two population means are equal ($\mu_1 = \mu_2$). A two-sample KS test showed that the distributions of the ground truth and predicted projected areas were found to have a KS value of 0.117, and a p-value of 0.389. As the p-value is large, there is no strong evidence to indicate

that the two distributions are not the same.

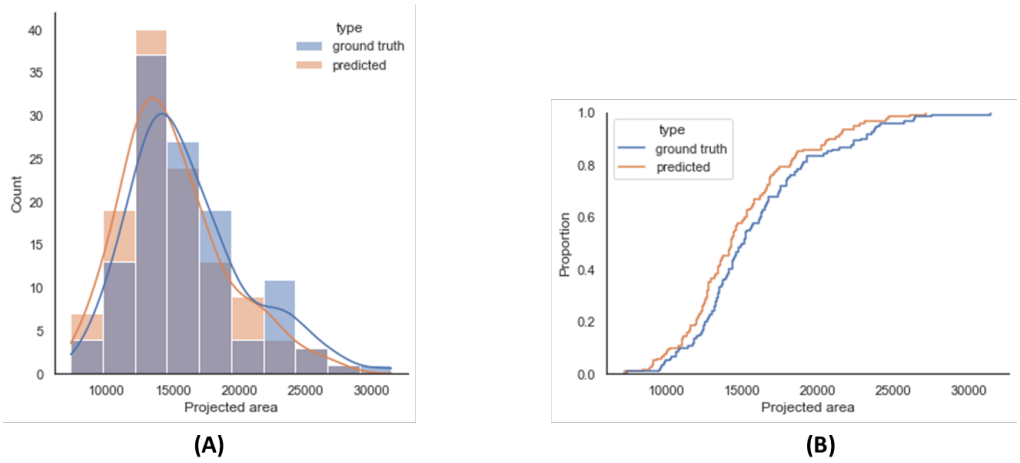

Figure 6.8: *Comparisons of the predicted projected area and the projected area according to the hand-labelled ground truth masks are shown using (A) histograms with KDEs overlaid and (B) ECDFs. Projected area is given as the pixel area, based on a* 1024×1024 *px image shape. While the shapes of the KDEs and ECDFs are very similar, the predicted projected areas are shifted to the left of the ground truth predicted areas, indicating that the predictions tend to under-report the projected area.*

In Section 6.1, Model 4 was found to be the most accurate pineapple detector based on validation AP values. As such, it was used to detect pre-measured pineapples in images that had not been previously seen by the model, as described in Section 6.2. The diameter, length and projected area were extracted from the masks detected by Model 4, using the OpenCV library. The values were compared to the calliper measurements (for diameter and length) and to the area of the hand-labelled ground truth masks (for projected area), using two-sample Z-tests and KS tests. These statistical methods showed that the distributions of the predicted measurements closely resembled those of the actual measurements. Hence, it appears that the proposed method is suitable for use in pineapple size determination from images.

Chapter 7

Conclusions, limitations and future work

This chapter provides an overview of the research conducted and summarises the findings and implications thereof. It also outlines the limitations of this research, as well as future work that could extend this project.

This research presented an approach to determine pineapple size from images, using Mask R-CNN to identify the instances of pineapples, and subsequently extracting fruit dimensions using the OpenCV library. All the research objectives mentioned in Chapter 1 were addressed. The development of a non-destructive pineapple fruit size determination approach based on images will be valuable in allowing the factory to obtain representative size data. This, in turn, will allow for a better understanding between the relationship between fruit size and juicing efficiency. This means that the factory will be better positioned to incentivise farmers to deliver the optimally-sized fruit by including a size-related factor into the pricing scheme.

Several Mask R-CNNs were trained on the pineapple dataset described in Section 5.2. Analysis of transfer learning showed that Model 2 (COCO_NoAug_Res101), initialised with MS COCO features, performed better than the Model 1 (Imagenet_NoAug_Res101), which was initialised with ImageNet weights. Both ResNet101 and ResNet50 CNN backbones were considered for the Mask R-CNN pineapple detector. It was found that Model 3, with the smaller ResNet50 backbone, performed almost as well as its larger counterpart.

The pineapple detector Model 3 (COCO_NoAug_Res50) achieved a satisfactory performance, with a validation AP@[0.50:0.05:0.95] of 0.884 and a test AP@[0.50:0.05:0.95] of 0.874. This was, however, improved upon by including data augmentation. Model 4, which made use

of horizontal flipping during the training process (COCO_Fliplr_Res50), achieved a validation AP@[0.50:0.05:0.95] of 0.914 and a test AP@[0.50:0.05:0.95] of 0.901.

From the observed error cases, it appears that the models perform better when the pineapples appear in a monolayer on the conveyor belt. Additionally, the camera heights could be adjusted more finely to ensure that both cameras have the same field of view. Alternatively, a crop and resize step could be employed on images from Camera B to enforce this.

Model 4 (COCO_Fliplr_Res50), having been identified as the best-performing detector, was then used to predict masks for pineapples in the previously-unseen dataset of pineapple images (described in Section 5.3), to post-validate the size determination approach. The distributions of the predicted fruit dimensions were found, using two-sample Z-tests and two-sample the Kolmogorov–Smirnov tests, to be equal to the manually measured fruits. It was therefore established that this is an appropriate method to use for pineapple size determination, in this context.

While the results achieved in this work were satisfactory, there were a few limitations:

- A relatively small dataset was used to train the Mask R-CNNs to detect pineapples. Additionally, these were acquired during a limited time period using convenience sampling, due to delays associated with the COVID-19 pandemic. Utilising training data obtained throughout the year might provide a detector that is more robust to variation in seasonal colour changes.
- Cameras were not installed at exactly the same height from the conveyor. If these heights were adjusted, the size determination accuracy would be increased.
- In the post-evaluation of the size determination approach, two-sample tests had to be used to compare the manual measurements to those extracted from the predicted masks, as it was not possible to identify individual pineapples in order to obtain a root mean square error (RMSE). It could have been useful to see how well the size determination approach worked with respect to individual fruits.

While convenience sampling was used in this work, future work could employ an experimental design approach, incorporating data from different seasons, as well as night shifts. Furthermore, the training data could be extended to include examples of foreign objects that are sometimes deposited onto the conveyor belts along with the fruit. Foreign objects may damage the peeler, resulting in downtime that negatively affects operating efficiency. If this could be implemented on a real-time basis, an auto-stop function could be incorporated to avoid damage to equipment. In future work, the size data obtained from images could be

used in conjunction with information about the growing conditions of the pineapple plants to better understand the factors affecting fruit size, and to allow for more accurate yield predictions.

Bibliography

[1] José Blasco, Nuria Aleixos, and Enrique Moltó. Machine vision system for automatic quality grading of fruit. *Biosystems engineering*, 85(4):415–423, 2003.

[2] GP Moreda, J Ortiz-Cañavate, Francisco Javier García-Ramos, and Margarita Ruiz-Altisent. Non-destructive technologies for fruit and vegetable size determination–a review. *Journal of Food Engineering*, 92(2):119–136, 2009.

[3] Jednipat Moonrinta, Supawadee Chaivivatrakul, Matthew N Dailey, and Mongkol Ekpanyapong. Fruit detection, tracking, and 3d reconstruction for crop mapping and yield estimation. In *2010 11th International Conference on Control Automation Robotics & Vision*, pages 1181–1186. IEEE, 2010.

[4] A Koirala, KB Walsh, Z Wang, and C McCarthy. Deep learning for real-time fruit detection and orchard fruit load estimation: Benchmarking of 'mangoyolo'. *Precision Agriculture*, 20(6):1107–1135, 2019.

[5] United Nations. Economic Commission for Europe and United Nations. Economic Commission for Europe. Working Party on Agricultural Quality Standards. *UNECE Standard FFV-49 concerning the marketing and commercial quality control of pineapples*. United Nations Publications, 2017.

[6] Food and Agriculture Organization of the United Nations. *Standard for pineapples CXS 182-1993*. United Nations Publications, 2011.

[7] J. Neibauer and E. Maynard. *Produce Quality Safety Information for Growers*. Dept. of Horticulture and Landscape Architecture, Purdue University, 2002. URL https://www.hort.purdue.edu/prod_quality/.

[8] J. A. Samson. *Tropical fruits*. Longman Scientific and Technical, 2 edition, 1986.

[9] Jun S Shin, Won S Lee, and Reza Ehsani. Postharvest citrus mass and size estimation using a logistic classification model and a watershed algorithm. *Biosystems engineering*, 113(1):42–53, 2012.

[10] Julia F. Morton. *Pineapple*, pages 18–28. Creative Resource Systems, Inc., 1987.

[11] DM Sether and JS Hu. The impact of pineapple mealybug wilt-associated virus-1 and reduced irrigation on pineapple yield. *Australasian Plant Pathology*, 30(1):31–36, 2001.

[12] VENCES C Valleser. Planting density influenced the fruit mass and yield of 'sensuous' pineapple. *International Journal of Scientific and Research Publications*, 8(7):113–119, 2018.

[13] José Naranjo-Torres, Marco Mora, Ruber Hernández-García, Ricardo J Barrientos, Claudio Fredes, and Andres Valenzuela. A review of convolutional neural network applied to fruit image processing. *Applied Sciences*, 10(10):3443, 2020.

[14] Yawei Wang and Yifei Chen. Fruit morphological measurement based on three-dimensional reconstruction. *Agronomy*, 10(4):455, 2020.

[15] S Dutta Gupta and Yasuomi Ibaraki. *Plant image analysis: fundamentals and applications*. CRC Press, 2014.

[16] Nicolai Häni, Pravakar Roy, and Volkan Isler. A comparative study of fruit detection and counting methods for yield mapping in apple orchards. *Journal of Field Robotics*, 2019.

[17] Ulzii-Orshikh Dorj, Malrey Lee, and Sang-seok Yun. An yield estimation in citrus orchards via fruit detection and counting using image processing. *Computers and Electronics in Agriculture*, 140:103–112, 2017.

[18] Antonio Ramón Jiménez, Anil K Jain, R Ceres, and Jose L Pons. Automatic fruit recognition: a survey and new results using range/attenuation images. *Pattern recognition*, 32(10):1719–1736, 1999.

[19] Sapan Naik and Bankim Patel. Machine vision based fruit classification and grading - a review. *International Journal of Computer Applications*, 170(9):22–34, 2017.

[20] Nanyang Zhu, Xu Liu, Ziqian Liu, Kai Hu, Yingkuan Wang, Jinglu Tan, Min Huang, Qibing Zhu, Xunsheng Ji, Yongnian Jiang, et al. Deep learning for smart agriculture: Concepts, tools, applications, and opportunities. *International Journal of Agricultural and Biological Engineering*, 11(4):32–44, 2018.

[21] Juliana Freitas Santos Gomes and Fabiana Rodrigues Leta. Applications of computer vision techniques in the agriculture and food industry: a review. *European Food Research and Technology*, 235(6):989–1000, 2012.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[23] Suchet Bargoti and James Underwood. Deep fruit detection in orchards. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3626–3633. IEEE, 2017.

[24] Inkyu Sa, Zongyuan Ge, Feras Dayoub, Ben Upcroft, Tristan Perez, and Chris McCool. Deepfruits: A fruit detection system using deep neural networks. *Sensors*, 16(8):1222, 2016.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[26] P Ganesh, K Volle, TF Burks, and SS Mehta. Deep orange: Mask r-cnn based orange detection and segmentation. *IFAC-PapersOnLine*, 52(30):70–75, 2019.

[27] Yuanyue Ge, Ya Xiong, and Pål J From. Instance segmentation and localization of strawberries in farm conditions for automatic fruit harvesting. *IFAC-PapersOnLine*, 52 (30):294–299, 2019.

[28] Jianxin Wu. Introduction to convolutional neural networks, 2016.

[29] Jim R Parker. *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.

[30] Scott E Umbaugh. *Computer imaging: digital image analysis and processing*. CRC press, 2005.

[31] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[33] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[34] Andrew Trask. *Grokking deep learning*. Manning Publications Co., 2019.

[35] Robert B Fisher, Toby P Breckon, Kenneth Dawson-Howe, Andrew Fitzgibbon, Craig Robertson, Emanuele Trucco, and Christopher KI Williams. *Dictionary of computer vision and image processing*. John Wiley & Sons, 2013.

[36] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016.

[37] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

[38] Chen Kong and Simon Lucey. Take it in your stride: Do we need striding in cnns? *arXiv preprint arXiv:1712.02502*, 2017.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[40] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[42] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[43] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[44] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *Neural networks*, chapter 11, pages 389–416. Springer series in statistics New York, 2001.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[46] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[47] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning. *Cited on*, 14(8), 2012.

[48] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[49] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2):261–318, 2020.

[50] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[52] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[53] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[54] Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. What makes imagenet good for transfer learning? *CoRR*, abs/1608.08614, 2016. URL http://arxiv.org/abs/1608.08614.

[55] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[56] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986. ISBN 0070544840.

[57] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[58] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[59] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[60] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[61] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[62] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.

[63] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2): 154–171, 2013.

[64] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[65] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[66] VideoLan. Vlc media player, 2006. URL https://www.videolan.org/vlc/index.html.

[67] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[68] Abhishek Dutta and Andrew Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi: 10.1145/3343031.3350535. URL https://doi.org/10.1145/3343031.3350535.

[69] Ekaba Bisong. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. Springer, 2019.

[70] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017.

[71] Francois Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras.

[72] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[73] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[75] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[76] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. https://github.com/aleju/imgaug, 2020. Online; accessed 01-Feb-2020.

[77] Regina L Nuzzo. Histograms: A useful data analysis visualization. *PM&R*, 11(3):309–312, 2019.

[78] George R Terrell and David W Scott. Oversmoothed nonparametric density estimates. *Journal of the American Statistical Association*, 80(389):209–214, 1985.

[79] *Parametric Test*, pages 412–412. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_307. URL https://doi.org/10.1007/978-0-387-32833-1_307.

[80] Douglas C Montgomery and George C Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010.

[81] *Central Limit Theorem*, pages 66–68. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_50. URL https://doi.org/10.1007/978-0-387-32833-1_50.

[82] David Lane, David Scott, Mikki Hebl, Rudy Guerra, Dan Osherson, and Heidi Zimmer. *Introduction to statistics*. David Lane, 2003.

[83] Les Underhill and Dave Bradfield. *Introstat*. Juta and Company Ltd, 1996.

[84] Jean D Gibbons and John W Pratt. P-values: interpretation and methodology. *The American Statistician*, 29(1):20–25, 1975.

[85] Raul HC Lopes, ID Reid, and Peter R Hobson. The two-dimensional kolmogorov-smirnov test. 2007.

[86] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[87] Michael A Stephens. An appreciation of kolmogorov's 1933 paper. Technical report, STANFORD UNIV CA DEPT OF STATISTICS, 1992.

[88] John L Hodges. The significance probability of the smirnov two-sample test. *Arkiv för Matematik*, 3(5):469–486, 1958.

[89] Herbert Büning. Robustness and power of modified lepage, kolmogorov-smirnov and crame´ r-von mises two-sample tests. *Journal of Applied Statistics*, 29(6):907–924, 2002.