EN3400 – Individual Project

Litter Detection with Computer Vision

Author: Thomas Rowland

Student Number: C1909002

Degree Programme: Electrical and Electronic Engineering

Academic Year: Year 3

Supervisor(s): Dr Yulia Hicks

CARDIFF
UNIVERSITY
PRIFYSGOL
CAERDYDD

EN3024 / EN3400 PLAGIARISM DECLARATION

I hereby declare:

that except where reference has clearly been made to work by others, all the work presented in this report is my own work;

that it has not previously been submitted for assessment; and

that I have not knowingly allowed any of it to be copied by another student.

I understand that deceiving or attempting to deceive examiners by passing off the work of another as my own is plagiarism. I also understand that plagiarising the work of another or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings against me.

Signed ....

Date 29/04/2022 ....

Abstract

Litter has become a part of our daily lives. It can be found in nearly every environment on the planet having damaging affects on the wildlife and our health. With the resurgence of progress within the field of deep neural networks, machine learning could play a part in alleviating this global litter epidemic. This project explores the use of deep learning and computer vision techniques, specifically object detection and classification for the purpose of litter detection. As progress is made in fields like artificial intelligence and robotics this problem will surely be one that is tackled. This paper introduces some background theory into how the neural networks are trained and implemented, highlighting some requirements for the system to work well in pictures and in live video. Some of the problems that come from trying to detect a class of objects as abundant and variable as litter is discussed in depth. Finally the project attempts to produce results within this field with the creation of a new litter dataset and multiple training configurations culminating in a classification precision of 75% and a mAP_0.5 of 36.1%.

Table of Contents

1 Introduction

1.1 The Litter Problem

As a species, we have become so desensitised to the sight of litter that we often no longer notice it. It is prevalent in all environments from the streets in London to the beaches of Cornwall and far beyond when it is washed into the oceans. It is estimated by Symphony Environmental that in the United Kingdom that 2.25 million items of litter are dropped every single day [1]. A survey by Keep Britain Tidy [2] conducted between April 2017 to March 2018 found the top 5 types of litter were smoking-related, confectionery wrappers, non-alcoholic drink containers, fast food-related and alcoholic drink-related across 7,200 sites of mixed-use. Non-alcoholic drink containers, which includes plastic bottles were found on 52% of the sites and it is estimated that over 700,000 plastic bottles are littered daily [3]. Larger items of litter such as plastic bottles, drinks cans, take-away containers and plastic bags or highly visible items such as crisp packets act as ‘beacons of litter’ [4]. These beacons act as an environmental and social cue, implying an increased social acceptability towards litter which then increases the amount of all types of litter dropped. These therefore would be valuable items to target in any clean-up attempt.

Litter pollution in the waterways is also a significant problem. According to the canal and River Trust [5] an estimated 14 million pieces of plastic rubbish end up in UK canals and river each year, leading to 500,000 pieces flowing into the oceans from UK waterways. An estimated 80% of ocean rubbish is first discarded on land [3]. Wind and rain wash these items into drains and waterways where they flow out to sea. Once there they are significantly harder to retrieve.

Yearly official national statistics are published detailing how much is spent by local authorities clearing litter. Between the year 2018-2019 the total cost was £699 million [6] in England alone, highlighting that litter poses a significant financial cost as well as environmental.

For the reasons stated above this project aims to explore the use of artificial intelligence for the autonomous detection of items of litter in any environment.

1.2 AI for Litter Detection

Object detection, which is a subfield of computer vision, is a technique that allows the detection and localisation of an object within a picture or video. It has made substantial advances in the last decade. The history of object detection can be separated into two periods, traditional methods [7, 8, 9, 10], pre-2014, which utilised hand-crafted methods to detect features and post-2014, where deep learning-based detection became the state-of-the-art. Traditional detectors like VJ Det [8, 9] and HOG Det [10] had more specific purposes. The VJ Det was designed for the detection of human faces within an image [9] and HOG Det [10] was primarily for pedestrian detection. Both of these cases have been adapted for other uses since, and have been an important foundation leading to many other object detection algorithms [7].

Alex Krizhevsky et. al. [11] pioneered the use of convolution neural networks (CNN's) for object classification in the ImageNet Large Scale Visual Recognition Challenge in September 2012. This development re-energised the research and 2 years later it was adapted by Ross Girshick et. al. [12] for the purpose of object detection on the PASCAL VOC dataset which is a dataset of over 11.5k images over 20 categories. Their use of CNNs in [12] improved the mean average precision (mAP) by more than 30% from previous methods.

It remains a highly researched topic and it is still a significant challenge within computer vision and AI. Many vision tasks that we require for real-world applications like autonomous driving, medical imagery, robot vision and video surveillance are object detection problems.

Section 2 will look at some background theory regarding machine learning, including some history with a brief overview into the current state of the art. It will also introduce the topic of deep learning and provide information on how neural networks utilise errors to improve. Lastly, an in-depth explanation of convolutional neural networks will illustrate the processes that lie between an input image and the output detection and classification of objects.

Section 3 will look at datasets. They are the driving force behind a data-driven system such as this and are the key to success or failure. Some different examples of notable object detection datasets will be discussed followed by an explanation of the different types of annotation within those datasets. A comparison of the available trash datasets will show the advantages and disadvantages of each for this project.

The “You Only Look Once” object detection algorithm will be introduced in section 4. YOLO is the chosen architecture for this project for reasons that will be explained, the section will give a brief explanation of the overarching aim of YOLO and a small run through of the different version improvements leading to the current version, YOLOv5.

A run through the work done in this project will be explained in section 5. Here a thorough rundown of the actions taken to firstly create a litter detector, and then iteratively searching for improvements in the accuracy of the system. This will include the cloud computing system used, the script, the creation of a new dataset and the analysis of the results.

Finally, this report will culminate in a discussion of the results of our investigation and potential future projects this might lead to.

2 Background Theory

2.1 Machine Learning

Machine learning algorithms are designed in a way to allow the computer to learn from experience [13, p.6]. The experience is taken as an input and is normally put forward in the way of a dataset. The algorithm then aims to identify and extract useful patterns within the input data, returning a model with the patterns encoded on it [13, p.6]. A simple example of this would be with the MNIST dataset [14], which is a dataset of 60,000 handwritten numerical digits 28 by 28 pixels in size. Figure 1 shows some example data from MNIST. This labelled data is fed into the program as shown in Figure 2 and it can be seen that the grayscale information is represented as an integer in the range of 0-255 with the whiter pixels correspond to a larger value. For greyscale

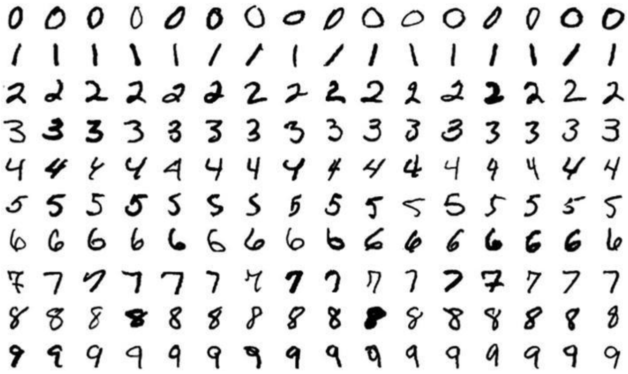
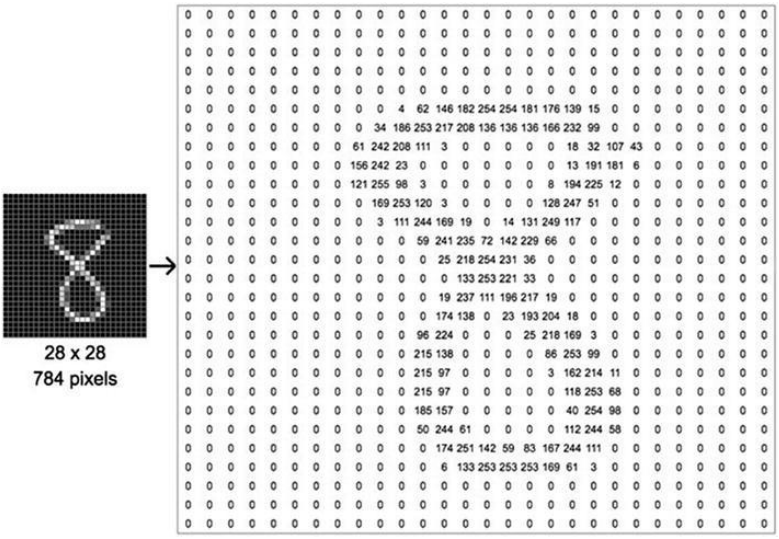


Figure 1 - Example data from the MNIST dataset. This figure was reproduced from chapter 3 of [15].

images such as those in the MNIST dataset a 2-dimensional matrix [Height, Width] is sufficient to pass across the image information. When a colour image is passed in a 3rd dimension is required [Height, Width, 3] to pass through the R-G-B image data [15].



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2 - An example of how the number 8 is represented in a matrix. Figure reproduced from chapter 3 of [15].

From this information the machine learning algorithm deduces the pattern of pixels that make up each digit and with many examples can encode this pattern onto the model [15]. This encoded model should be able to generalise to other examples of the training data that it hasn't seen yet and has therefore 'learnt' to recognise the numbers.

This process becomes a lot more complicated as the complexity of the image increases and even with this simple example it become clear that it would be extremely difficult for a programmer to manually code a program to do this.

2.2 Neural Networks

A neural network as the name suggests is a computational model that is inspired by the brain - albeit an extremely simplified version. The network is made up of a large collection of information processors termed 'neurons'. The first neural network, called Perceptron [15], shown in Figure 3, was not much of a network as it only contained one neuron however a neurons real value in modern neural networks does not come from individual neurons but the cumulative result of many.

2.2.1 Perceptron

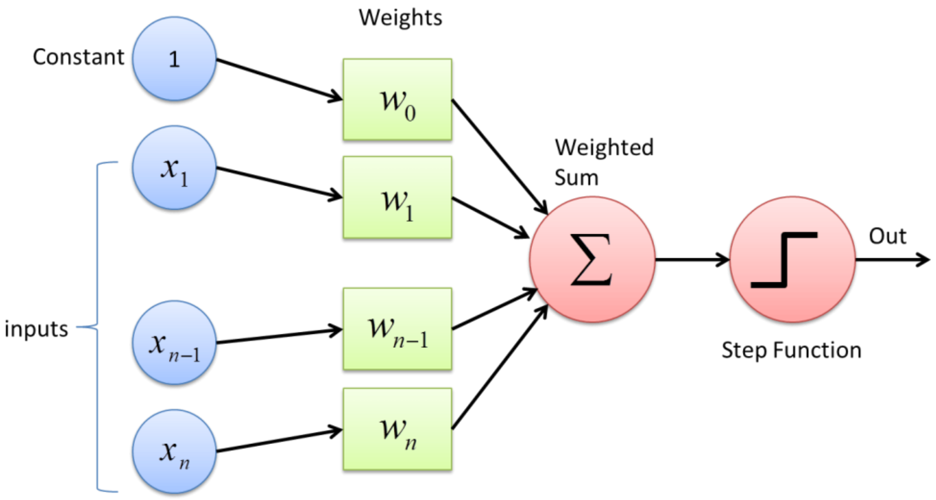


Figure3 - A diagram showing the single-layer neural network, Perceptron. Reproduced from [16]

The process for a single neuron is simple as shown above in Figure 3. The inputs to the neuron get multiplied by a weight. The resulting values then get summed along with a bias and then passed through an activation function, which in the case of perception is a unit step function. Written mathematically the first process within artificial neuron can be described as [13, p.72]

$$z = \sum_{i=1}^{n} (x_i \times w_i) \qquad (1)$$

z = neuron output

x_i = neuron input

w_i = neuron weight

The activation function provides nonlinear mapping to the weighted sum, in the case of perceptron it was a threshold function / step function. These were common in early neural network research [13, p.73].

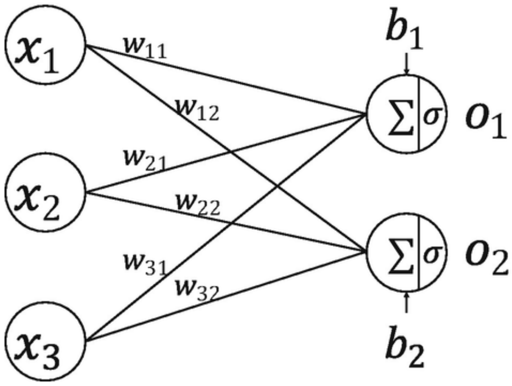


Figure 4 - Two perceptrons in parallel in a fully connected layer. Reproduced from chapter 3 of [15].

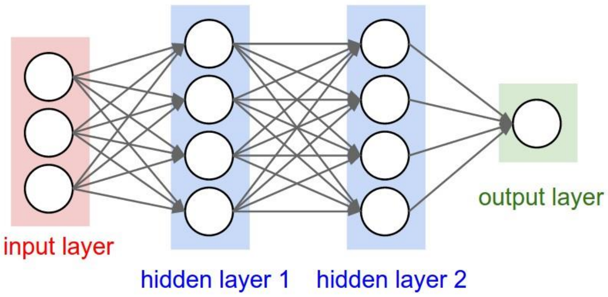


Figure 5 - A fully connected 3-layer neural network. Reproduced from [17].

The diagram of perceptron can be expanded to show two perceptrons in parallel in Figure 4, and a 3-layer neural network as shown in Figure 5. In Figure 5 you can see the input layer outputs to the first layer of neurons, these output to a second layer which then outputs to some output neurons. Figure 4 illustrates how the weights, also known as parameters, are split between the layers. The weights are initialised to random values and through the process of training the neural network on the dataset, the weights are slowly adjusted to improve the accuracy on that data as described in section 2.3.

2.2.2 Activation Functions

The activation function is part of the second stage of the neuron. The function is used to map the weighted sum unput of the neuron to a specified output depending on the neurons/units task within the network. As mentioned previously, the threshold activation function was common in early neural network designs and can be described as follows [13, p.74]:

$$threshold(z) = \begin{cases} (z \geq 0) \rightarrow +1 \\ (z < 0) \rightarrow -1 \end{cases} \tag{2}$$

Nowadays these are not commonly used as they do not allow for the fine-tuning of neurons. The Rectified Linear Unit or ReLU is the most popular activation function in modern neural network architectures [13 p.73]. The ReLU is described below.

$$threshold(z) = \begin{cases} (z > 0) \rightarrow z \\ (z \leq 0) \rightarrow 0 \end{cases} \tag{3}$$

The activation functions of neurons can be different, but generally the activation function is kept uniform throughout a layer [13, p.79].

2.2.3 Biases

The bias is added to the calculation to adjust the value of the weighted sum prior to being passed into the activation function [13, p.88]. This is analogous to the '+c' term in the equation of a line and will move to shift the neurons decision boundary a certain way. With the activation and bias included the equation describing output of a neuron now looks like:

$$NeuronOutput\ =\ ActivationFunction(\sum_{i=1}^{n}(x_i \times w_i)\ +\ bias) \qquad (4)$$

2.3 Deep Learning

A 'deep' neural is a neural network more anything more than 2 layers. The layers are counted from the first hidden layer to the output layer, the input layer is not counted as it does not any weights associated with it. Deep neural network architectures can vary from 2 layers to many thousands [19]. The real benefit of using deep learning comes from utilising the many non-linear layers for the extraction of features from the input data.

2.4 Learning functions

What makes a neural network stand out from other strategies is its ability to improve. Training a neural network really means the process that the network goes through with the training data to determine the set of weight values that best model the patterns in that data [13, p186]. It can normally be described as initialising the weights to some random value and iteratively adjusting the weights according to the errors that the network has made on the dataset. Two commonly used training algorithms for this process are the Gradient Descent and Backpropagation algorithm.

2.4.1 Gradient Descent

The gradient descent algorithm is an optimization algorithm. It uses the error between the model's prediction and the truth to determine the quickest way to update the weights to minimise the error. The most common way for the loss to be determined is with the sum squared errors (SSE) [13, p.191]. The equation describing the SSE is as follows:

$$SSE = \frac{1}{2}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2 \tag{5}$$

n = dataset examples

y_j = correct target value for dataset example j

$\hat{y}_j$ = model estimate of target value of dataset example j

As the model estimate is a product of the weights at that neuron and the input of the neuron the above equation can be expanded to form Equation 6 [13, p.201]:

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \left(\sum_{i=0}^{m} w_i \times x_{j,i}\right)\right)^2 \tag{6}$$

From this, the error gradient is determined by taking the partial derivative with respect to the weights. These equations are as follows but for simplicity the notation from Equation 5 is used to represent the models estimate [13, p.202]:

$$\frac{\partial SSE}{\partial w_i} = \sum_{j=1}^{n}\left((y_j - \hat{y}_j) \times -x_{j,i}\right) \tag{7}$$

As previously discussed, the inner bracket represents the output of the weighted sum and the " $- x_{j,i}$" represents the rate of change of the weighted sum with respect to the change in w_i. The negative can be dropped from the rate of change as we are interested in the downhill gradient in the descent to the minimum.

This can then be used to define the gradient decent weight update rule in Equation 8, which shows by how much each weight should be updated.

$$w_i^{t+1} = w_i^t + \left(\eta \times \sum_{j=1}^{n} \left(\left(y_j^t - \hat{y}_j^t \right) \times x_{j,i}^t \right) \right) \tag{8}$$

w_i^{t+1} = next weight value
w_i^t = current weight value
η = learning rate

Points to note about the above equation, due to the input weight in Equation 6 the equation will generalise to all weights and they will be updated proportional to the individual error gradient. This means that the optimisation will not take the most direct route and will curve towards the minimum. The learning rate η is a parameter used to moderate the learning speed so that too larger steps cannot happen in a given iteration [13, p.204].

Gradient descent as it has been described so far will only work for a network comprised of a single layer. In datasets with a large number of examples it is computationally very costly to try and run through this process with every example before every slight adjustment. To combat this stochastic gradient descent is employed. The above algorithm sees a random sample of examples from the dataset, normally anything between 1 and several hundred, with the default being 32. Once the weight adjustment has been made another random sample is used. To be useful to a deep neural network it must be used in combination with another algorithm, most commonly the backpropagation algorithm.

2.4.2 Backpropagation

The backward propagation algorithm is a method that can be used to update the weights of neural network with hidden layers. There are two phases to it, the forward pass and the backward pass. In the forward pass, an input is passed to the input neurons of the network, the activations of each neuron flow through the network to the output later, an example of which is shown in the 3-layer network in Figure 5. The weighted sum calculations seen in Equation 1 and the activations are stored in memory [13, p.211]. The overall error for the whole network is calculated at the output layer by comparing the model’s prediction with the truth from the data example. It is likely that initially, the error will be high as the weights are initialised to random values. Now the challenge comes when trying to assign blame for that error to each individual neuron and change it accordingly. This is achieved in the backward pass and gradient descent. The error gradient for each of the output neurons, often referred to as $\delta\ (delta)$, [13, p.215] will be calculated from Equation 7. The error gradients are then backpropagated from the output layer to their inputs which are the neurons of the last hidden layer. This process repeats through every layer until each neuron receives a portion of the blame for the overall network error in proportion to how the output of that neuron effects the network. This will show a vanishing gradient as the δ decreases as the backpropagation passes back through the network. Once the error values have been established then a weight optimisation strategy like that explained above can be used to update the weights. This gets repeated through every training example until the networks error converges to a level where it is either acceptable or it won't converge any further.

2.5 Convolutional Neural Networks

Convolution neural networks (CNN's/ConvNets) were originally designed for image recognition and applied to recognising handwritten digits such as those described in section 2.1 [13, p160]. Their aim is for the early layers within the model to act as feature detectors and extract local features with the latter layers to be dedicated to combining these features to form higher-order features.

2.5.1 Feature Extraction

The architecture of ConvNets are different from regular neural networks. Regular neural networks utilise a series of fully connected 2D hidden layers, where the neurons of that layer are connected to every single neuron of the previous layer, ending with a final fully connected output layer. ConvNet layers have 3-dimensions, height, width and

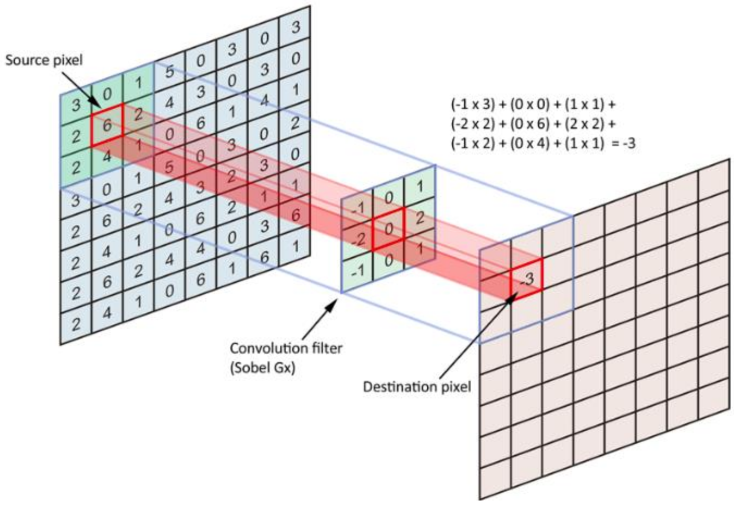

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 6 - An illustration of how the convolution filter adds values to the feature map from the 2-dimensional input image matrix. Replicated from [20]

depth, which in the case of colour images is 3 (R-B-G). Layers are not fully connected either, only a small region is passed from one layer to the next [21]. The input image, which is a matrix like that in Figure 6 has a convolution filter (often called a kernel) passed over it which then produces a feature map. The area of the filter is called the receptive field and the dimension sizes will generally be odd numbers. Figure 6 shows how the convolution filter performs matrix multiplication with the values in the input image matrix to produce one value on the feature map on the right. With different filter values, different features will be detected from the input image and therefore multiple filters are used and added to multiple feature maps.

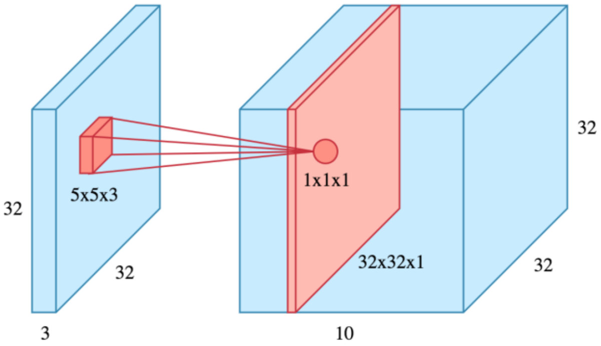


Figure 7 - An illustration of the convolution of a 3-dimensional input image matrix into 10 feature maps. Figure reproduced from [20]

Figure 7 shows how the convolution operation on a 32x32x3 input image with 10 filters produces a 32x32x10 feature map. The height and width of the feature map are preserved due to padding, which is where extra columns and rows are added to the outside of the input matrix, commonly filled with zeroes [20]. Adding an extra column and row of zeros in the input image in Figure 6 would allow you to get a value for [0,n] and [n,0] in the feature map. The convolution filter is passed from left to right and then down 1 row, analogous to reading in the English language. This movement by a certain step size called the stride. The step size can be varied to miss matrix elements if only looking for high level features, this will also reduce the size of the feature map [20].

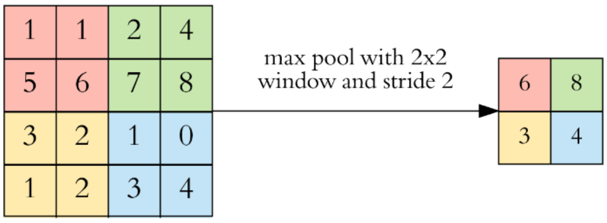


Figure 8 - An example of max pooling on a feature map. Replicated from [20]

Due to the high computation requirements required to carry out matrix multiplication on large matrices pooling layers are used after a convolutional layer to reduce the dimensions of the feature map. Reducing these computations will reduce the training time of the network and can prevent overfitting [20]. One example of pooling which is the most common is max pooling. A window of a specified dimension is placed on the feature map and the max value forms the element value in the next matrix. A visual example is shown in Figure 8. As you can see, max pooling with a 2x2 window has decreased the dimension size and the number of weights by a factor of 4 whilst still preserving the important information. ConvNets often have millions weights, so this can be a substantial computational saving [20]. With multiple filters, pooling does not affect the depth, only the height and width of the feature map.

After the convolution and pooling layers the network will contain some fully connected layers [20]. As shown in figures 4 and 5 the hidden layers require a 1-dimensional vector of values instead of the 3-dimensional result from the previous layers. The values get ‘flattened’ into a 1D vector. To again use the reading analogy, it would be the equivalent to putting a page of text into one long line. Each line of elements gets concatenated onto the end of the previous.

2.5.2 Classification

The fully connected layers after the convolutional layers are what enables the classification of any detected features. The earliest convolutional layers may detect an edge, then later layers may build on those to form a shape. For a system trained to detect multiple classes a softmax activation function will be used to predict the probability of each class being in the image. The equation for the softmax activation function can be seen below.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{n=1}^{K} e^{z_n}} \tag{9}$$

$\vec{z}$ = input vector

K = number of classes

e = exponential function

The results of equation 3 will leave one class with the highest probability and that will be the networks classification. To visualise what the output at the softmax layer the keras-vis module [22] can be used to get the network to generate an image that it thinks represents a class the most. Figure 9 shows the generated representations from 12 classes replication from [20].

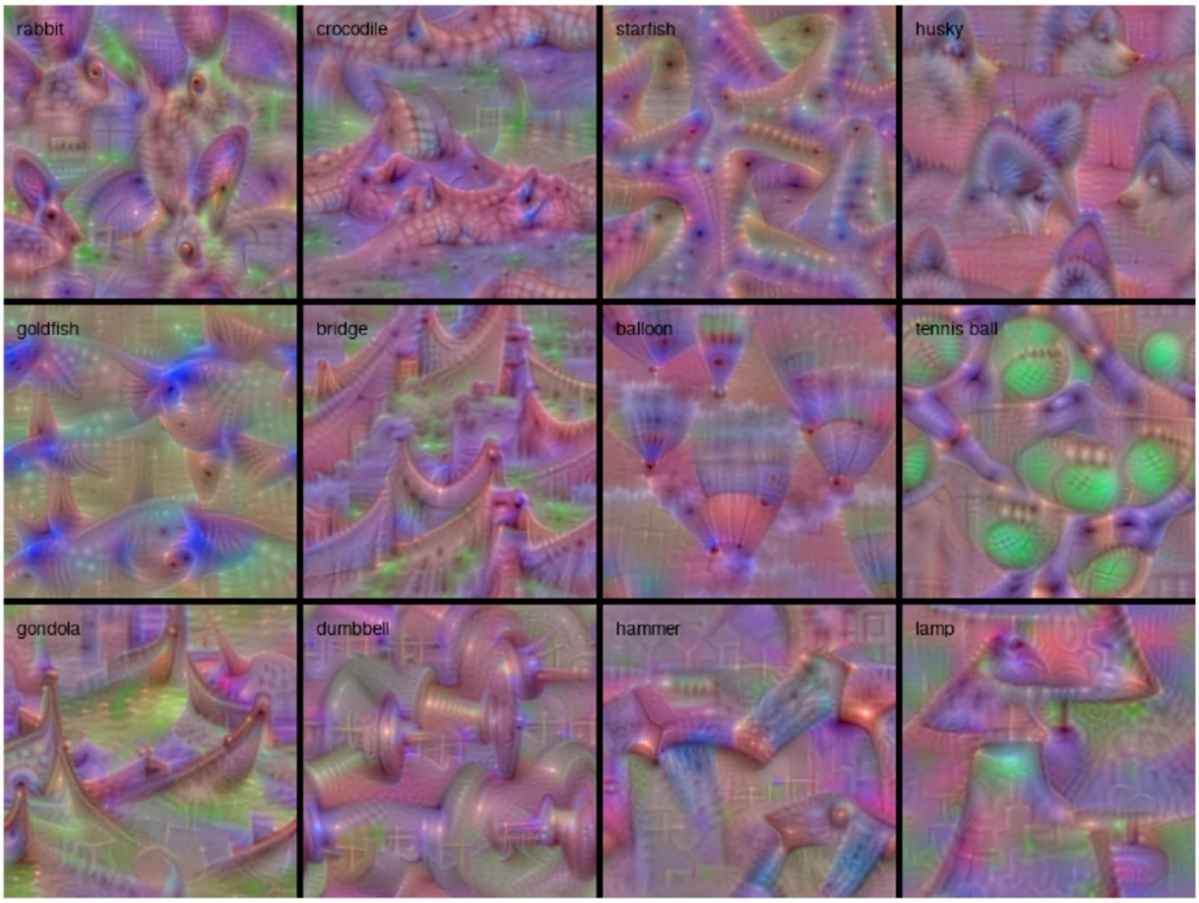


Figure 9 - The output at the softmax layer visualised by showing a networks representation of certain classes. Reproduced from [20]

3 Datasets

As previously mentioned in section 2.1 and then highlighted in Figure 9, the driving force behind the success of machine learning and deep learning is the ability for the system to extract features from the data and not those previously programmed by the designer. This collection of data is called a dataset and for the purpose of this project it will consist of a series of annotated images of litter. Other forms of machine learning applications may require a different type of dataset, for example a dataset for a natural language processing project (NLP) could be a collection of spoken words such as the Noisy Speech Database [23] or a set of labelled SMS messages such as the “SMS Spam Collection” [24] which is a dataset of nearly 6000 spam SMS messages.

The success of a project can largely depend on the quality of the dataset. Dataset design and preparation can take up a considerable amount of time and resources in a project. In some cases approaching 80% of a projects budget can be spent on these measures [13, p. 32], [25].

3.1 Training, Validation, Testing

The test set is split into 3 separate parts: train, val and test. The training set is the images that get passed through the model when the weights are being adjusted. These are the images that the model generates the loss gradients from and will amend the weights accordingly. The validation set of images are still used during training [26], but they do not affect the weights. The model is tested on these images at the end of each epoch (when the model has seen all of the train images) and they are just used to get metrics evaluating the performance of the model. They could signal that there would be no more performance gains from more training or that the model is performing poorly which might require an early stop. The test set is used at the end of the training and is a set of data that the model had never seen before. As the validation set is still used during the training it is beneficial to get a better sense of the models performance with new data. A benchmark split between these datasets is about 70%, 20%, 10% [26].

3.2 Dataset Examples

Datasets for training an object detection system will mostly come in the form of annotated images or video. Some notable examples of these datasets are ImageNet [27], CIFAR10 [28], Microsoft COCO [29] and Open Images [30]. All are image datasets. ImageNet, CIFAR-10 and MS-COCO have been attributed to making significant advances in computer vision and a lot of networks will have been pre-trained on these datasets to get the low-level features.

Table 1 - A comparison of image datasets commonly used in computer vision.

Name	No. Instances	No. Categories	Task	Year
ImageNet	14 million	20,000	Detection	2009
CIFAR-10	60,000	10	Classification	2009
MS-COCO	2.5 million	80	Classification and detection	2015
Open Images	16 million	600	Classification and detection	2017

The above table, Table 1, contains a comparison of the named datasets along with some of the attributes that make them useful for computer vision tasks. With the exemption of CIFAR-10 they are all created for object detection. The CIFAR-10 dataset is different as it contains very small images (32 x 32 pixels) where the object fills a lot more of the whole image and there is only 1 instance per image. Some examples of the CIFAR dataset can be seen below in Figure 10.

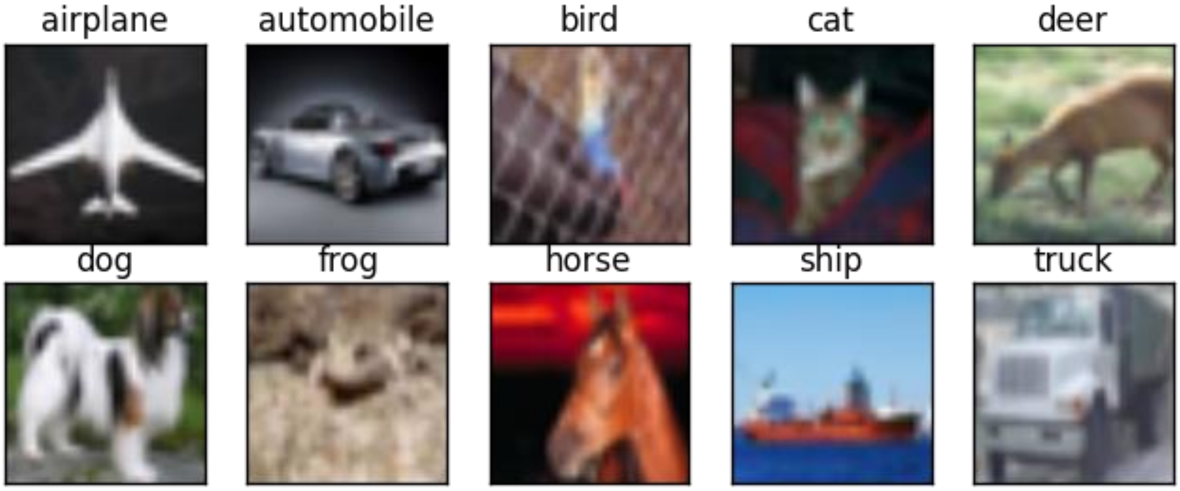


Figure 10 - Examples of the images contained within the CIFAR-10 dataset – Replicated from [26]

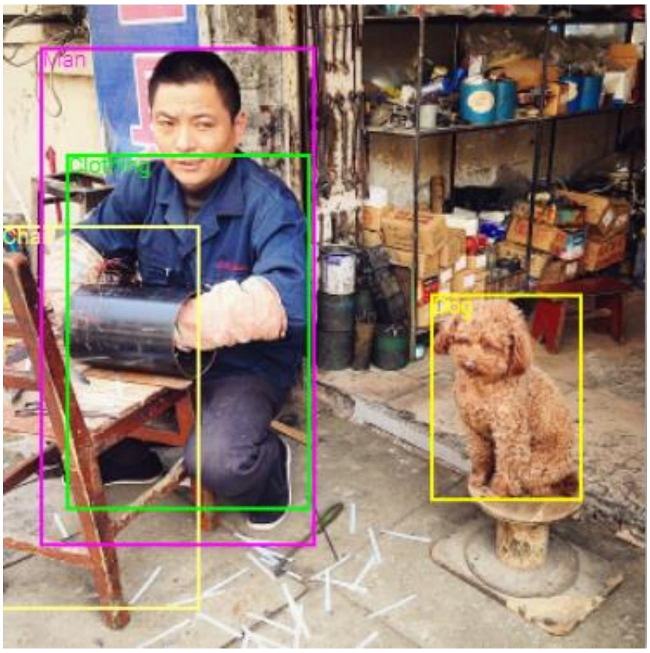


Figure 11 - An example from the Open Images dataset [30] illustrating multiple instances in one image

In comparison to Figure 10 above, Figure 11 shows an example from the Open Image dataset. Here you can see that there are multiple different categories in one image. Each object instance is surrounded by a bounding box to serve as a point of reference for that object.

3.3 Image Annotation Types

Different objectives may require a different style of image and annotation. For most computer vision tasks there are four annotation types to choose from, image classification, object detection/recognition, segmentation and boundary recognition.

3.3.1 Image Classification

An example of this type of data is shown in Figure 10. Each image contains only 1 object and the of the system is to train the program to recognise an object in an unlabelled image. Its role is to only identify the presence of an image and has no interest in any location information.

3.3.2 Object Detection

Figure 11 is an example of this type of annotation. The annotation information is intended to show the presence, number, and location of any objects within the image. As shown in Figure 11, multiple different classes can be labelled or there could be multiple instances of the same class.

3.3.3 Segmentation

This can be further split down into semantic segmentation, instance segmentation and panoptic segmentation. Figure 12 shows examples of these styles of annotation.

Semantic segmentation will provide the presence, location and size and shape of an item if it is stand-alone. It doesn't not discriminate between similar objects and will label them under the same identification [32] as show in Figure12(b).

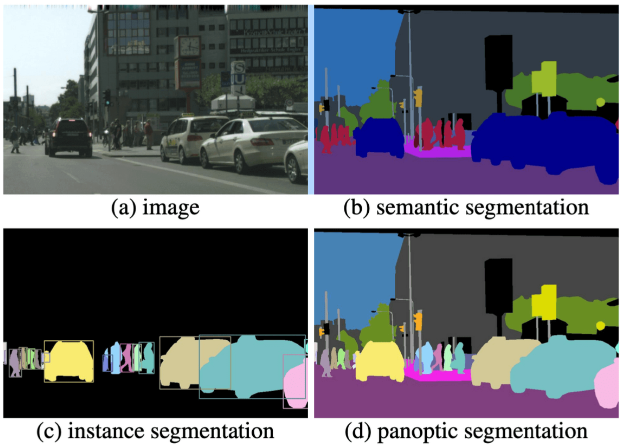

(a) image
(b) semantic segmentation
(c) instance segmentation
(d) panoptic segmentation

Figure 12 - Examples of segmentation types. Replicated from [32]

Instance segmentation as the name suggests goes a step further and will always display the size and shape of an object. It will show the presence, location, count, size and shape of any object within the image.

Lastly panoptic segmentation is a blend of semantic and instance segmentation. It will provide information for the objects and the background.

3.3.4 Boundary Recognition

This type of annotation is used to train the computer to recognise boundaries of an object within an image. This can be utilised by systems trying to distinguish the road from the pavement for example. As this method is not directly relevant to this project no more will be said on it. For more information on this annotation type see “Holistically-Nested Edge Detection” [33].

3.4 Dataset Annotation Techniques

Different object detection models may require a different annotation format. As has been discussed in previous section the annotation is used to provide information about the object. Two relevant techniques for this annotation are the bounding box, as seen in Figure 11, and the polygon as demonstrated in Fig.9. Both styles have their advantages. The bounding box style is significantly quicker to do the annotations, the polygon on the other hand is a lot more accurate as model recognises only the actual object to be the class, not the object and the background enclosed in the bounding box.

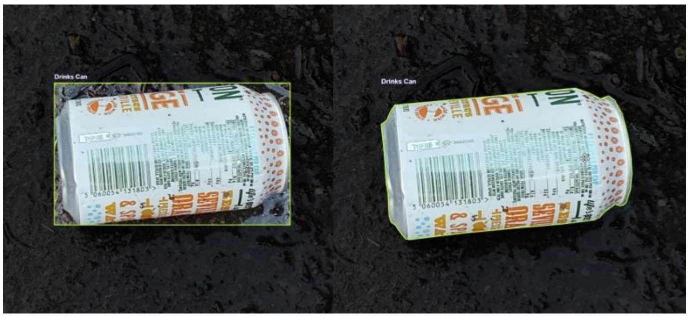


Figure 13 - Examples of both the bounding box annotation style on the left and the polygon annotation style on the right.

3.5 Trash Datasets

Using object detection for the purpose of identifying litter is not a new idea. There have been previous projects looking to tackle similar issues such as: specifically waste detection in the oceans [34, 35, 36, 37], categorising waste that is flowing under a bridge in a river [38] or using a drone to quickly categorise litter in a larger area [39]. From these and other studies there are several open source litter datasets to choose from as seen in Table 2.

Table 2 - A comparison of available open source waste datasets.

Name	No. Categories	No. Subcategories	Number of Images	Annotation type	Comment
TrashNet	6	-	2,527	Classification	Clear Background
TACO	28	60	1,500 (+3500)	Segmentation	Waste in the wild
Trash-ICRA19	3	24	5,700	Detection	Underwater
UAVVaste	1	-	772	Segmentation	Drone dataset
TrashCan 1.0	3	34	7,212	Segmentation	Underwater

The above table, Table 2, shows some attributes of some of the available open source litter datasets. The table includes, TrashNet [40], TACO: Trash Annotated in Context [41, 42], Trash-ICRA19 [43], UAVVaste [44, 45] and Trashcan 1.0 [46, 47]

Requirements of a dataset in this project ideally would be that they are similar to the conditions for which the program would be utilised in. Training on Trash-ICRA19 and TrashCan 1.0 would still be of some benefit as the system can still learn to recognise items than are found both in the ocean and on the sea floor such as bottles and carrier bags. Due to the images being taken under water, the image quality is not as good and there are few waste categories. The UAVVaste dataset has images of waste in the wild, but they are taken from a drone, so the objects are small with respect to the image size. TrashNet is a good option, but as the objects are taken in front of a white background it is not as representative of the task the program is designed for. The TACO dataset is the best option as it contains 1500 images of sufficient quality in the environment that we are aiming to deploy to. The image annotations have been outsourced to regular users via the portal on the dataset website [41] and checked by the dataset creator to make sure that they are accurate. There are a potential 3500 other annotated images that have not been checked. See appendix A for a breakdown of the number of annotations per category in the verified dataset.

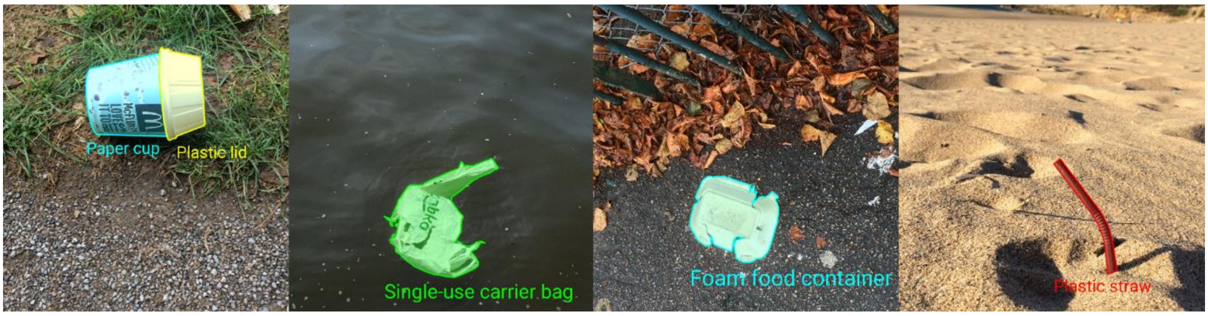


Figure 14 – Four examples of annotated images from the TACO project. Reproduced from [41].

4 Litter Detector

The neural network architecture used in the TACO project [42] was the Mask R-CNN [48]. The Mask R-CNN is a framework designed for object detection. It provides high quality instance segmentation with a segmentation mask overlay, like those seen in Fig.14. Initially the project plan was to replicate this TACO project in using the Mask

R-CNN and think on ways to improve or compare with other architectures. It was decided another framework would be better. This was due to lack of experience training neural networks, the fact that the Mask R-CNN is not compatible for testing on a CPU [42] as it is computationally heavy during inference and that the delay makes it a less than real-time detector at only 5 fps. You Only Look Once (YOLO) [45] was chosen as a better architecture for the project as there are more available resources online and it will function for testing on a CPU. Detection speeds are also more than sufficient for real-time detection with speeds up to 140 fps with a GPU. It also will be more deployable to compute limited devices beyond this project.

4.1 You Only Look Once (YOLO)

YOLO has become a series of object detection models that offer real-time detection at a reasonable accuracy. The initial version, YOLOv1 was released in a paper titled “You Only Look Once: Unified, Real-Time Object Detection” by Joseph Redmon et. al in 2015 [49]. Prior to this paper object detection in images utilised repurposed image classifiers where approaches such as the R-CNNs first generate bounding boxes in likely regions of interest are and then run an image classifier on those proposed bounding boxes. This double jointed approach made for higher computation and less than real time inference speeds. YOLO uses a single convolutional neural network that simultaneously predicts bounding boxes and predicts class probabilities for that box [49]. This allows YOLO to see the entire image during training and testing which means it implicitly learns contextual information about classes, as well just the appearance of that class. YOLO divides an input image to an S × S grid. Responsibility

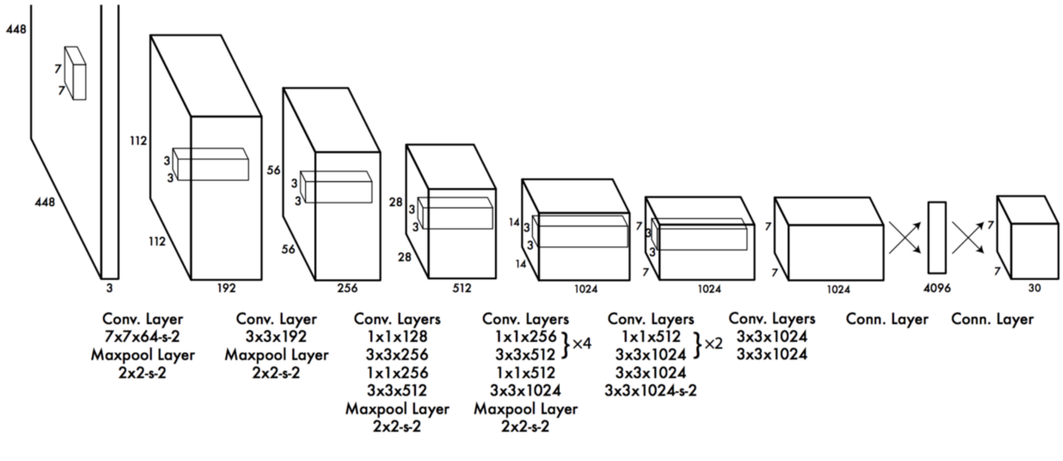


Figure 15 - YOLOv1 Architecture. Reproduced from [CITE]

falls on the grid square containing the centre of an object for its detection. Every square predicts bounding boxes and associates a confidence score with it that encompasses both how likely it is that the bounding box contains an object, and how accurate the bounding box it likely to be around the object.

Version 1 contained 24 convolutional layers for the feature extraction followed by 2 fully connected layers for the prediction out of the output probabilities and coordinates. Leaky rectified linear activation functions (Leaky ReLU) were used throughout with the exception of the final layer which used a linear activation function. The leaky ReLU can be described by [49]:

$$\sigma(z) = \begin{cases} (z > 0) \rightarrow z \\ (z \leq 0) \rightarrow 0.1z \end{cases} \tag{10}$$

YOLOv2 [50] was released in 2017. Created by the same author, it made some iterative improvements [51] to YOLOv1 such as BatchNorm to convolutional layers (re-centreing and rescaling) and higher resolution in the classification layers. In 2018 YOLOv3 [52] made improvements to the bounding box predictions and made class predictions at multiple granularities to improve detection of smaller objects. The original author of YOLOv1/2/3 Joseph Redmon stepped away from computer vision research and further improvements to the YOLO algorithm were carried out by other researchers. YOLOv4 [53] (2020) made improvements with small changes to the architecture and adding data augmentations. Data augmentations are modifications to the input images such as rotation and blur which effectively give you extra training instances.

4.1.1 YOLOv5

YOLOv5, the most recent iteration, was not released with any accompanying paper, instead it undergoes ongoing development on its repository by Ultralytics at [54]. The major difference between YOLOv5 compared with its predecessors is the framework. A machine learning framework is the software library that the code is written in. YOLOv1 through YOLOv4 used the Darknet Research Framework [55] whereas Glenn Jocher of Ultralytics translated and improved the training procedures into PyTorch [56]. The Darknet framework is primarily written in C and offers very fine control over the network operations. This can be helpful to certain research avenues, but it is a lot more complex and therefore slower to make progress. Pytorch on the other hand is available in Python and C++, making it a lot more familiar to a lot of programmers, it also allows the use of some popular python packages such as NumPy, SciPy and Cython. Other advantages are the ease of learning and easier debugging. YOLOv5 was also improved its data augmentation steps and now the training images get 3 augmentations by the algorithm, scaling, colour space adjustments and mosaic augmentation. Mosaic augmentations were a novel implementation and it combines four images into tiles of a random ratio. An example of the mosaic augmentation can be seen in appendix F. These all move to increase the variability of the training set and to increase the number of annotated images, effectively increasing the size of the training dataset.

YOLOv5 comes in different size variations. The table below copied from the Ultralytics repository shows the different model sized and inference speeds when trained on the COCO val2017 dataset.

Table 3 - A table displaying the inference speed of YOLOv5 models with a CPU and a GPU with a batch size of 1. Reproduced from [cite]

Model	**size**	**mAPval (%)**	**mAPval (%)**	**Speed (ms)**	**Speed (ms)**
	(pixels)	**0.5:0.95**	**0.5**	**CPU b1**	**V100 b1**
YOLOv5n	640	28	45.7	45	6.3
YOLOv5s	640	37.4	56.8	98	6.4
YOLOv5m	640	45.4	64.1	224	8.2
YOLOv5l	640	49	67.3	430	10.1
YOLOv5x	640	50.7	68.9	766	12.1

The above table shows that with a GPU for inference all the options are available for processing in video in near real time. However with just a CPU for testing there are only 2 options suitable for processing in near real time assuming that to be anything over 10 frames per second. That is the YOLOv5s and the YOLOv5n. For the increased mAP the YOLOv5s is the best option for this project and the vast majority of the testing will be carried out with that. The model has evolved throughout the iterations and YOLOv5s is now 270 layers with 7.06 million weights.

5 Testing and Training

The TACO project's [42] GitHub page [57] gives some information on how to move forward with utilising the dataset with their model. It explains that there are 1500 images that are part of the main dataset and there is a separate dataset of 'unofficial' images and annotations. These have not been checked by the author and should not be trusted completely as good annotations. Experience was gained using the command prompt to clone repositories, to download the dataset and to download and install the required modules. When attempting to run the detector.py program as a test on a laptop an error would halt the program due to the lack of GPU.

5.1 The use of GPUs in deep learning

The Graphics Processing Unit (GPU) has become a requirement for deep learning since breakthroughs in the late 2000s [13, p.153]. As discussed in section 2.2 and 2.4 a neural network can be thought of as a string of matrix multiplications separated by non-linear activation functions. Central Processing Units (CPU) can be used to train neural networks but when there is the potential to have millions of parameters in a neural network it is not feasible to use them. GPUs have been optimised to run many very fast mathematical calculations in in parallel and therefore can handle the multiplication of many parameters significantly quicker.

Initially the project plan had been to use the GPUs on the Hawk Supercomputer [58]. Due to the learning curve required to run projects with Linux on the Hawk system coupled with the learning curve using neural networks it was decided that a cloud computing platform such as Google Colab would be a good option for the initial testing and prototyping as there is more supporting literature and it has a more intuitive user interface.

5.2 Google Colab

Colaboratory is website from Google Research [59], it allows users to write and execute Python scripts in a browser without any configuration required. It provides free (but sometimes limited) access to GPU's or TPU's. A single .pynb file can be made on colab that allows the incorporation of text and images with the code into a single notebook, making it an excellent tool when starting out with machine learning. There are also some negatives that come with using Google Colab. A dataset will have to be uploaded onto the server every time it is used as uploaded files get deleted when the runtime finishes. This can cause a bit of a trouble should the user disconnect from the internet or close the browser as the files will have to be re-uploaded and the runtime restarted.

An alternative solution to load a dataset is to use the computer vision developer framework Roboflow [60]. This software allows the user to upload images and annotations, add pre-processing and augmentation steps to the images and store different dataset versions on their server. The software can also be used to annotate unannotated images in polygon or bounding box form and then export the annotated datasets in multiple formats, i.e. YOLOv5 Pytorch, PASCAL VOC or COCO.

5.3 Uploading TACO Datasets to Roboflow

Initial attempts to correctly upload the TACO datasets (both the checked and unchecked) despite Roboflow saying that they automatically convert the formats it was found that this was not the case. As you can see in Figure 16 the annotations did not match up to the images and some annotations where completely broken which meant the dataset could not be used at without solving this problem.

To solve this problem more research had to be done into the annotations of the TACO dataset, how they differ from the required format, how the annotation is mapped to the image and how the annotation file is formatted.

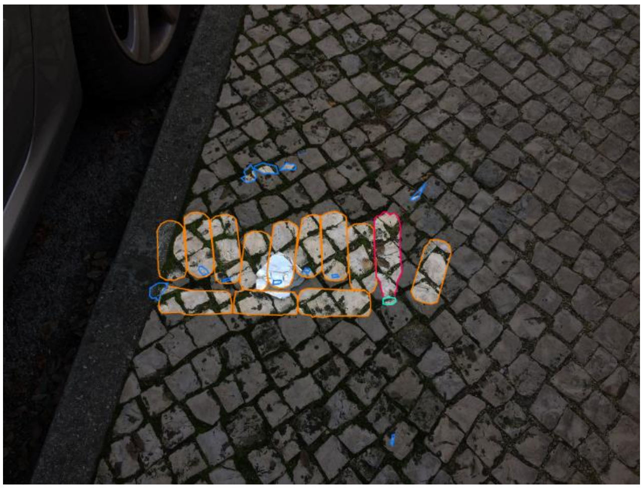
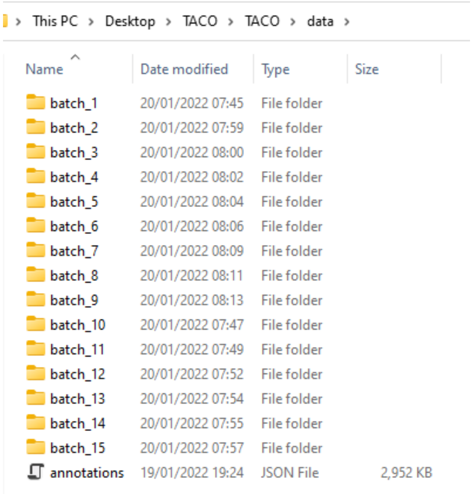


Figure 16 - An example of the annotations after initially uploading the TACO dataset to Roboflow.

5.3.1 COCO vs YOLO Annotation Format

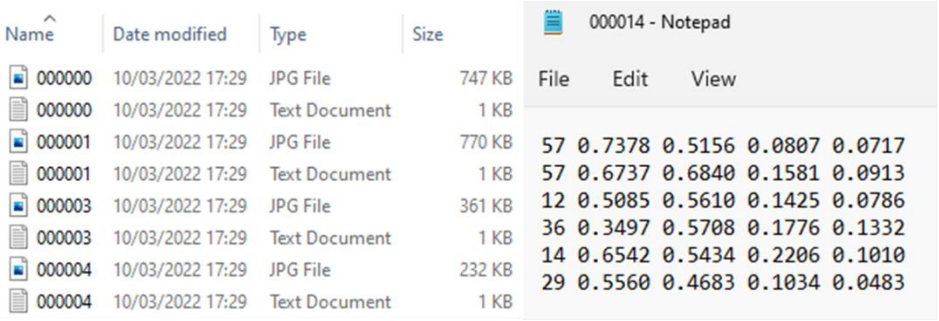
The TACO dataset download is in the COCO format and the model requires the YOLO format. There are several differences that must be addressed if the annotations are to be converted. Upon download the data directory is shown in Figure 17. In each of the batch folders there is just ~100 images and there is one annotations.json file. The annotations file contains all of the information associated with dataset. This includes the high-level information about the dataset, the license information, the image specific information with a complete list of every image name and dimensions, the annotation specific information and the category information. See appendix B for an example annotation file.



Name	Date modified	Type	Size
batch_1	20/01/2022 07:45	File folder	
batch_2	20/01/2022 07:59	File folder	
batch_3	20/01/2022 08:00	File folder	
batch_4	20/01/2022 08:02	File folder	
batch_5	20/01/2022 08:04	File folder	
batch_6	20/01/2022 08:06	File folder	
batch_7	20/01/2022 08:09	File folder	
batch_8	20/01/2022 08:11	File folder	
batch_9	20/01/2022 08:13	File folder	
batch_10	20/01/2022 07:47	File folder	
batch_11	20/01/2022 07:49	File folder	
batch_12	20/01/2022 07:52	File folder	
batch_13	20/01/2022 07:54	File folder	
batch_14	20/01/2022 07:55	File folder	
batch_15	20/01/2022 07:57	File folder	
annotations	19/01/2022 19:24	JSON File	2,952 KB

Figure 17 – In COCO format the annotations is contained within 1 file.

In comparison the YOLO annotation format uses a single text file for every single image. The text file only contains the specific object information. The category id and the four coordinated of the bounding box reference to the centre of the object.



Name	Date modified	Type	Size
000000	10/03/2022 17:29	JPG File	747 KB
000000	10/03/2022 17:29	Text Document	1 KB
000001	10/03/2022 17:29	JPG File	770 KB
000001	10/03/2022 17:29	Text Document	1 KB
000003	10/03/2022 17:29	JPG File	361 KB
000003	10/03/2022 17:29	Text Document	1 KB
000004	10/03/2022 17:29	JPG File	232 KB
000004	10/03/2022 17:29	Text Document	1 KB

```
57 0.7378 0.5156 0.0807 0.0717
57 0.6737 0.6840 0.1581 0.0913
12 0.5085 0.5610 0.1425 0.0786
36 0.3497 0.5708 0.1776 0.1332
14 0.6542 0.5434 0.2206 0.1010
29 0.5560 0.4683 0.1034 0.0483
```

Figure 18 - An example of how each image has its own annotation file and an example text file containing the category id and bounding box coordinate information.

A .yaml file must be made in the parent directory to correctly map the category id's to a category name. The .yaml file that was created to accompany the TACO dataset is in appendix C. It contains all of the category names as well as the path to the training and validation images.

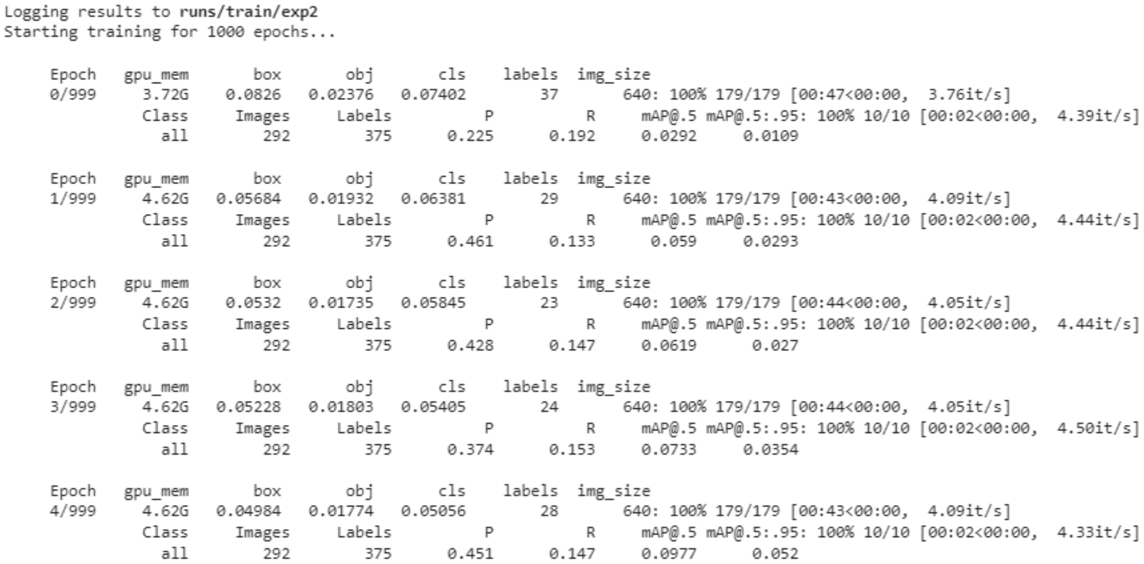
Lastly as to use the dataset a python script was used that extracts the data for each image from the annotations.json file, converts the annotation coordinates from COCO format to YOLO and creates the .txt file for each image. This script was run in google colab and can be seen in appendix D.

5.4 Script

The YOLOv5s training script is adapted from the Ultralytics repository for use with the custom litter training set. Due to the nature of cloud computing the start of the script is installing the dependencies, cloning this Ultralytics repository and connecting the runtime to external applications such as Roboflow to load the dataset in to Google Colab, and Weights and Balances for performance monitoring. Next the specific dataset iteration is loaded into the runtime from your Roboflow account into the test, val and test folders. The training program is called with the below snippet of code.

```
!python train.py --img 640 --batch 32 --epochs 500 --
data {dataset.location}/data.yaml --weights yolov5s.pt –cache
```

It is in this line command that the hyperparameters can be controlled and batch size and training length can be adjusted. When training starts information about the model is displayed the output window starts populating the results of training epoch as they come through as shown in Figure 19.



```
Logging results to runs/train/exp2
Starting training for 1000 epochs...

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     0/999     3.72G    0.0826   0.02376   0.07402        37       640: 100% 179/179 [00:47<00:00,  3.76it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 10/10 [00:02<00:00,  4.39it/s]
                 all        292        375      0.225      0.192     0.0292     0.0109

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     1/999     4.62G   0.05684   0.01932   0.06381        29       640: 100% 179/179 [00:43<00:00,  4.09it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 10/10 [00:02<00:00,  4.44it/s]
                 all        292        375      0.461      0.133      0.059     0.0293

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     2/999     4.62G    0.0532   0.01735   0.05845        23       640: 100% 179/179 [00:44<00:00,  4.05it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 10/10 [00:02<00:00,  4.44it/s]
                 all        292        375      0.428      0.147     0.0619      0.027

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     3/999     4.62G   0.05228   0.01803   0.05405        24       640: 100% 179/179 [00:44<00:00,  4.05it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 10/10 [00:02<00:00,  4.50it/s]
                 all        292        375      0.374      0.153     0.0733     0.0354

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     4/999     4.62G   0.04984   0.01774   0.05056        28       640: 100% 179/179 [00:43<00:00,  4.09it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 10/10 [00:02<00:00,  4.33it/s]
                 all        292        375      0.451      0.147     0.0977      0.052
```

Figure 19 - A snapshot of the output once training of the model has begun.

Upon completed of training the detect.py program is called to run inference of the test dataset with.

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 640 --
conf 0.1 --source {dataset.location}/test/images
```

The output from this code block displays inference speeds per image as well as the classes detected in each image. These images are then displayed for visual confirmation and the weights downloaded. When the session ends in Google Colab; which can be caused by manually stopping, loss of connected or a time-out disconnect, all of the files, including the weights that have been learnt through training will all be deleted. This makes it important to download the weights to local memory storage so that they can be used for future use.

5.5 Plastic Bottle Training Set

Whilst working on the conversion script as there was no working dataset it was decided that it would be a good idea to make a small practice dataset so experience in other parts of the project could be gained. Using pictures gathered from a google image search a small dataset of 51 images containing 60 plastic bottles and 42 plastic bottle caps were annotated using the annotation tool on Roboflow. This was the first time training on a custom dataset and upon initial examination it looked to be doing well.

The metric for determining the success of an object detection program is the Mean Average Precision (mAP) which looks at classification and localisation of the bounding box [61]. To understand what this means it should be broken down. The precision is the percentage of predictions in that are correct [62].

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{12}$$

This does not consider any missed objects within an image or how well the bounding box is localised over the object. For each prediction the area of the overlap between the predicted and the ground truth is measured, called the intersection over union (IoU).

$$Intersection\ over\ Union = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{13}$$

Or using proper notation:

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{14}$$

A threshold value is set for the IoU, this will determine whether the prediction is recorded as a true positive or a false positive [61]. This threshold value sweeps over a range of values Recall is a measure of how well the system finds the objects.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{15}$$

Average Precision (AP) can be found from the area under a precision-recall curve. The mAP is the mean of these values.

After training the model on the plastic bottle dataset for 280 epochs (training cycles) the following graphs were produced in the training monitor python package Weights & Biases (Wandb) [62].

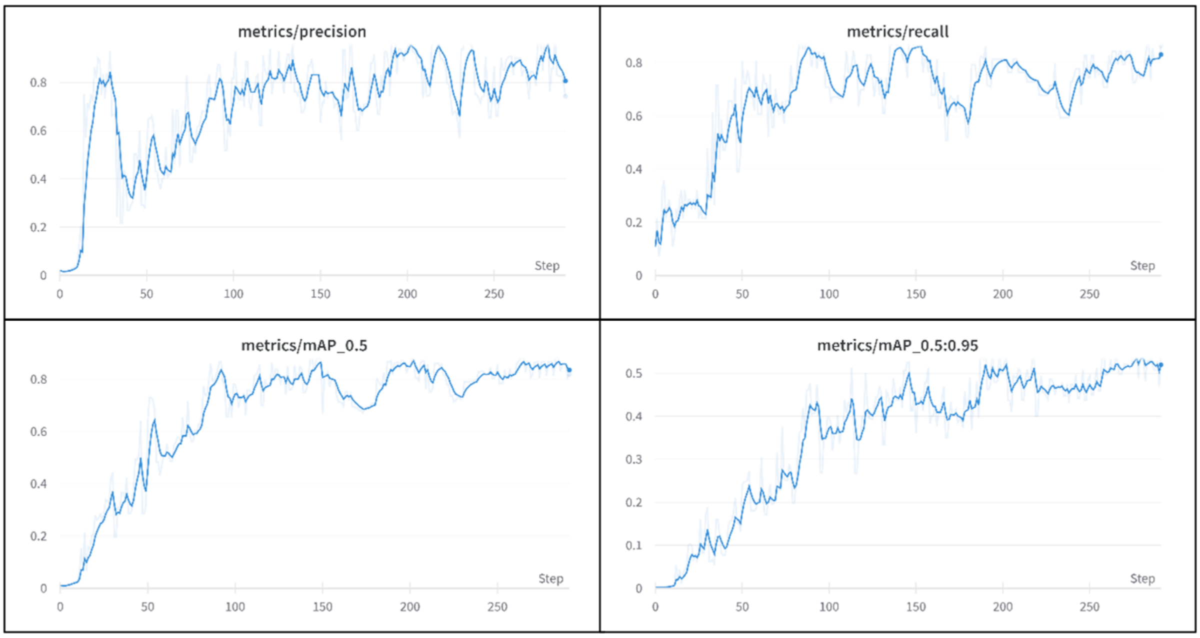


Figure 20 - A set of smoothed graphs showing precision metrics compared with training epoch for the test plastic bottle dataset.

The graphs show that for the first 100 epochs the precision increases and then levels off a bit. The bottom graphs show the average precision with a threshold value of 0.5 (left) and when the threshold value sweeps from 0.5 – 0.95 in 0.05 increments. In this instance the mAP would is described as [61]:

$$mAP = \frac{1}{|thresholds|} \sum\nolimits_{t} \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{16}$$

Storing the weights at the best observed epoch, which was 186, we can run the detection model on the test images in the dataset as shown below in Figure 14.

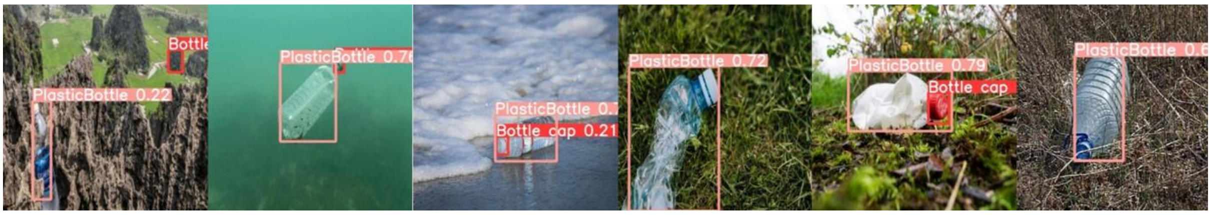


Figure 21 - Inference results on the test images in the plastic bottle dataset

The images above show that the model looks to have detected the plastic bottles with sufficient accuracy. There is one false positive in the image on the left with the system incorrectly labelling the rock as a bottle cap and a few false negatives where the program failed to identify the bottle caps in 3 of the images. The numerical value next to the classification label is the confidence score, which relates to the confidence in the classification and the localisation of the bounding box.

5.6 Initial TACO Training Runs

With the TACO dataset converted, there was a dataset with a potential dataset size of ~5000 annotated images to train with. The first attempt with a proper dataset used just used the 1500 verified images. The model performed very poorly as shown below.

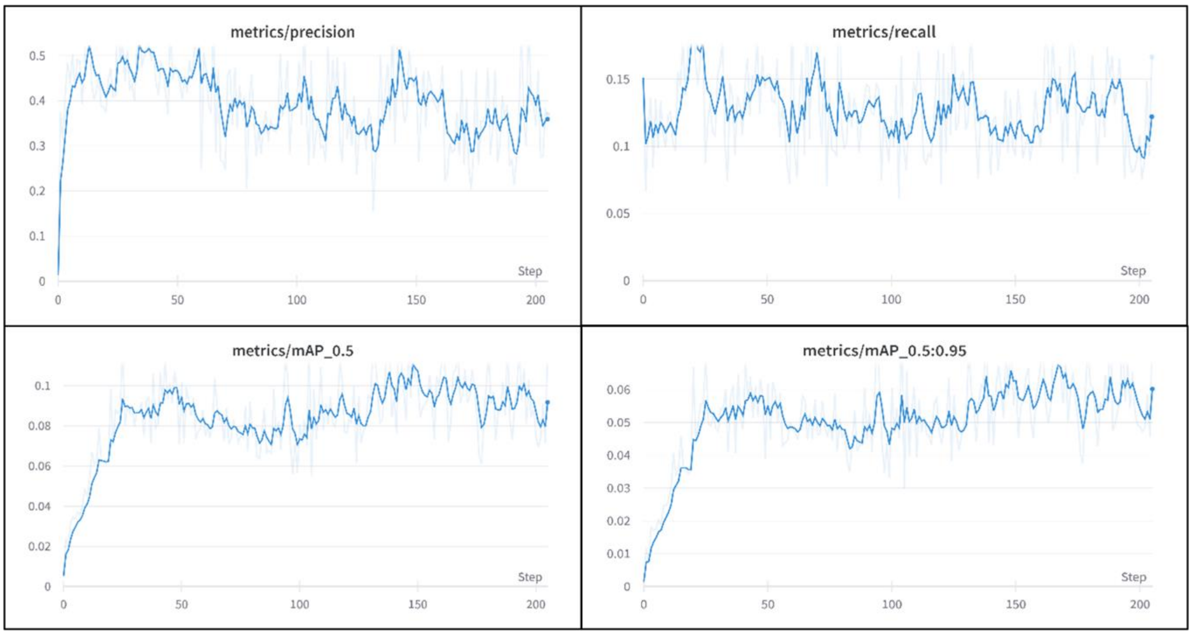


Figure 22 - The precision results from the first training run on the TACO dataset

The graphs in FIGURE 14 show the results to be significantly worse than those recorded with the plastic bottle dataset. The bottom left graph shows that less than

10% of the objects in the image are overlapped by a correctly classified bounding box by more than 50%. This is not sufficiently accurate for our model and there are many avenues to go down to look for improvements.

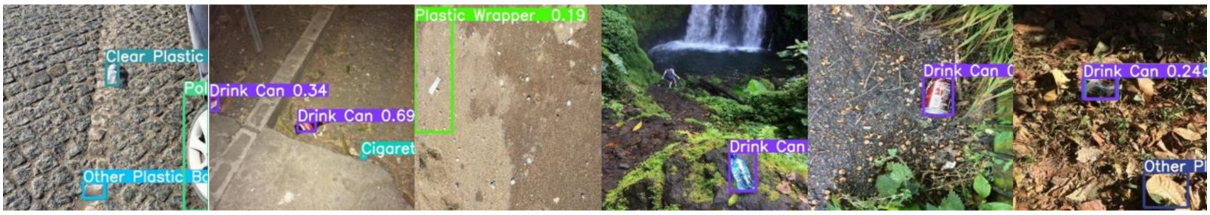


Figure 23 - A selection of examples from the TACO dataset test. The system accuracy is significantly worse

Initially it was thought that the dataset was not big enough for the task. 60 categories is high for a dataset with only 1500 images. Merging the checked and unchecked images together gives a total of 4923 images in the dataset. The results did not improve, much to the contrary with the mAP_0.5 settling at 0.08 or 8%.

5.6.1 Health Check

Roboflow has a function that it calls a "health check". This function shows how many annotations the dataset contains for each class. Using this on the verified TACO dataset showed there the be a large spread in annotation count between classes. This is unsurprising as items such as cigarettes are small and highly littered and an image can contain many whereas items such as a shoe or piece of Tupperware are not so much. This health check can be seen in appendix E. This meant that some classes were severely under-represented and some classes where very over-represented. Which in turn would mean the parameters within the network would be geared more to a cigarette detector than a general litter detector after training. Another line of thought was that many of the classes were too general, for example 'unlabelled litter', 'food waste' and 'other plastic'. These items can come in the form of many different shapes, sizes, colours and textures so the model would find it very hard to detect enough patterns in these classes to be able to generalise to other objects of the same class.

The severely under-represented classes were removed as well as the classes that were too general and the model was trained again. The mAP_0.5 improved to 0.165 with the mAP_0.5:0.95 improving to 0.106. A lot better than previous attempts but still short of anything that could be deployed.

5.6.2 Confusion between classes

Many of the classes may cause confusion with other classes due to similar appearance. There are several examples in this dataset that might cause some confusion including: plastic lid and metal lid, other plastic bottle and clear plastic bottle (especially when not empty), paper cup and plastic cup and drinks can and food can. Several training runs were carried out in various configurations to establish if this made any significant difference There were some improvements with mAP_0.5 topping out at about 0.2. There were large gains in precision with runs that only included up to 5 classes. See below in Figure 24 of the precision graphs when the model was trained on corrugated cardboard, normal white paper, drinks can pull tabs and carrier bags. The system is performing better but detecting just niche categories such as these isn't the purpose of this project.

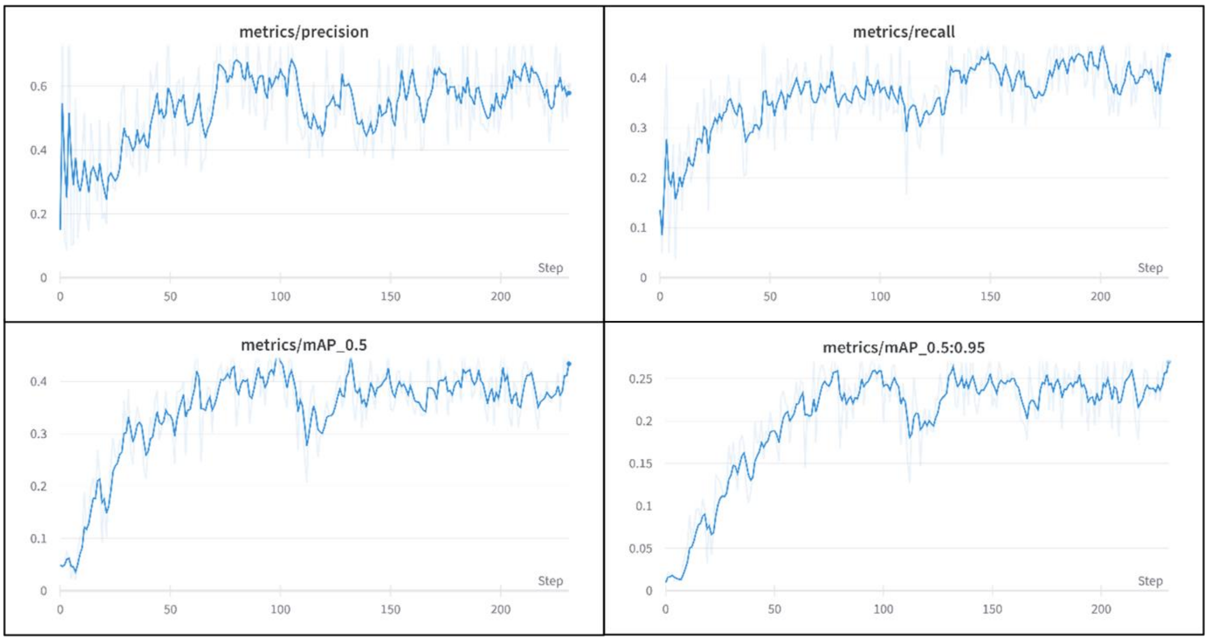


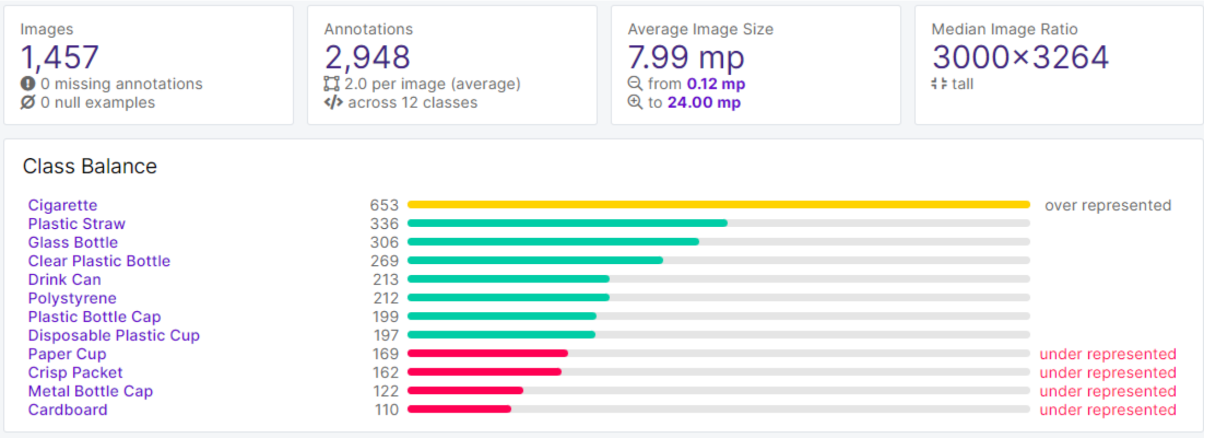
Figure 24 - Results from initial testing with only a 4 classes present

The system was still performing poorly when asked to categorise more than 5 classes and by handpicking some select classes that might have obvious uniform patterns such as drinks pull tabs seemed to detract from the aims too much and a different approach had to be thought up. As already highlighted in previous sections the quality of a machine learning system falls heavily of the quality of the dataset. This prompted an in-depth check of the images that were being used. It had been assumed that because the images from the TACO dataset had been checked by the author and that the initial checks after the conversion, that of the images would be of the same standard. This unfortunately was not the case. Some of the images were ambiguous and unrecognisable even to the human eye as to what category they fall in, or some were too far away or bad quality to be of any use. It was decided that a new dataset should be made which would focus on some select categories, contain images gathered from the TACO verified dataset, the TACO unverified dataset and 300 self-collected images.

5.7 New Litter Dataset

An aim of 15 rubbish categories was set for this new dataset and the decision of which categories would be based on a few factors. They would hopefully already contain a reasonable amount of annotated images in the TACO verified and unverified datasets as it is quicker to check and alter existing annotations than do them from scratch. They wouldn’t be too ambiguous so the items such as unlabelled litter and other plastic would be left out. Referring back to section 1.1 and [4], it is said that some larger or highly visible items of litter such as plastic bottles and crisp packets act as beacons of litter, these would be valuable categories to try and include. In the end 12 categories made the final dataset, some desirable categories such as carrier bags proved too general and time consuming to annotate.

This process took considerable time, but the end result was 2,948 annotation over 12 categories in a dataset of 1,457 images which had all been checked and annotated. The class balance was still uneven, but it was a lot better than before and can be seen below.



Images
1,457
0 missing annotations
Ø 0 null examples

Annotations
2,948
2.0 per image (average)
</> across 12 classes

Average Image Size
7.99 mp
from 0.12 mp
to 24.00 mp

Median Image Ratio
3000×3264
tall

Class Balance

Cigarette	653	over represented
Plastic Straw	336	
Glass Bottle	306	
Clear Plastic Bottle	269	
Drink Can	213	
Polystyrene	212	
Plastic Bottle Cap	199	
Disposable Plastic Cup	197	
Paper Cup	169	under represented
Crisp Packet	162	under represented
Metal Bottle Cap	122	under represented
Cardboard	110	under represented

Figure 25 - The class balance of the newly formed litter dataset

Cigarettes are the obvious outlier in the above data. They were included because it remains one of the most highly littered items in the world and the model should be tested to see how it performs with them included. However, the shape and size of cigarette butts do not make them suitable for detection as they often inhabit only a small number of pixels with respect to the whole image.

5.7.1 New dataset results

Results with the new dataset were initially disappointing. The average precision had improved but was still struggling to get over 0.25 mAP_0.5 meaning that only a quarter of the images where correctly annotated with the bounding box covering 50% of the object. Different augmentations such as flip and rotate were explored but as the yolov5 architecture already contains some of these they did not make much difference.

5.7.2 Confusion matrix

It was discovered that a useful tool to debug a poorly performing object detection model is the confusion matrix.

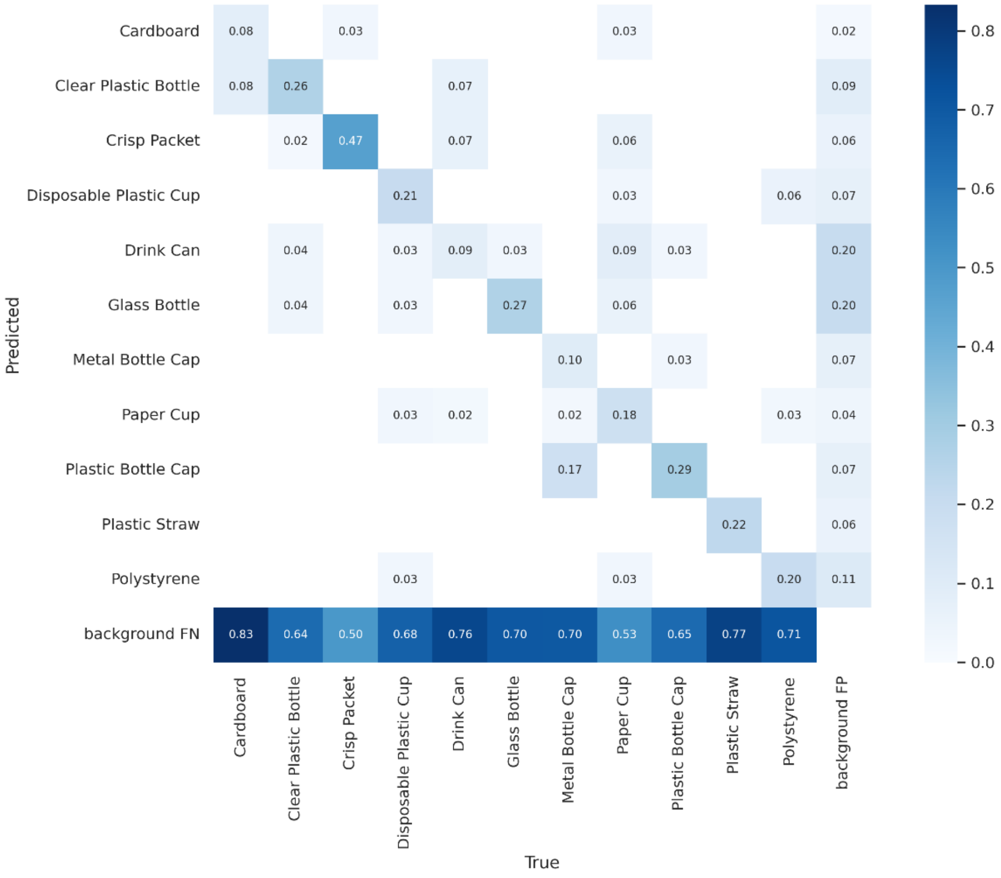


Figure 26 - A confusion matrix showing a run containing 11 classes on the YOLOv5s.

The figure above is an example of a confusion matrix that was produced from the second training run with the new dataset. This run contained all categories except cigarettes. A confusion matrix of a model that is performing well would have a dark line of blocks running from top left to bottom right. This would show that what the model is predicting an object to be is indeed what that object is. The confusion matrix above however illustrated that the most common confusion by far is with the background. The model mistakes an object for the background, either by annotating a random section as an object or by missing the detection completely.

This information was then confirmed by passing only the bounding box of each object into the model, in effect turning it from an object detection model to an object classification model. When faced with and image that contained only the object the model performed well with the mAP_0.5:0.95 reaching 0.75 as shown in Figure 27.

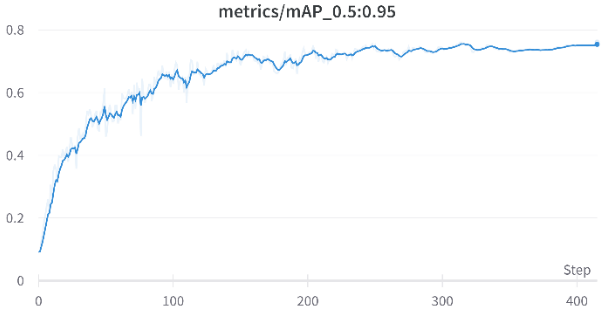


Figure 27 - The mAP_0.5:0.95 of 11 classes in the new dataset with just the bounding box passed into the network.

5.7.3 Size and Dimension changes

Upon realisation that the background was causing issues, in an effort to improve the performance a static crop was added to each image. An example of this the effect of cropping the image is shown below.

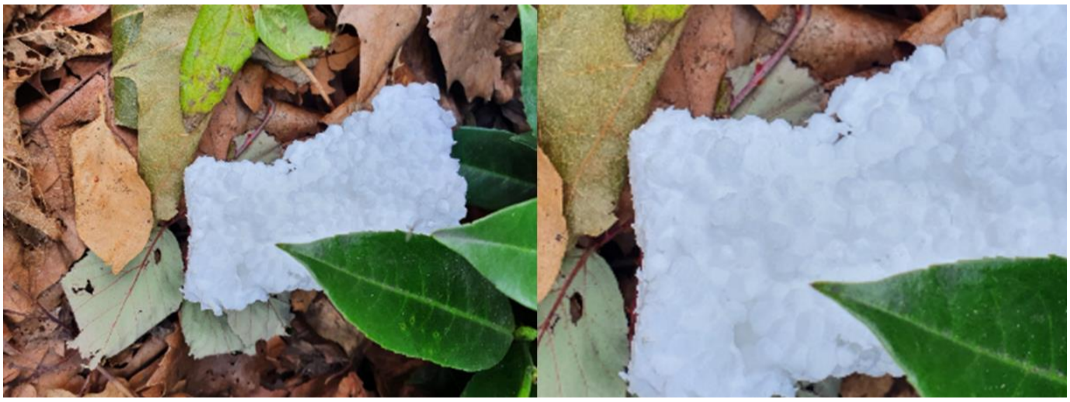


Figure 28 - An image showing the effects of a 25% crop on an image from the dataset.

This would artificially increase the size of the object whilst reducing the background if the annotation is away from the edges like it is in most of the images. There is some loss in details in some images as you can see above, some of the object has been lost. This change resulted in an improvement of the mAP_0.5 to 0.31 and the mAP_0.5:0.95 to 0.21.

The dimensions of the input images were also changed from 416x416 to 640x640 pixels which saw a further increase in precision. This did however increase the time taken to train the network. With all of the changes added the best final precision of the system detecting 11 classes was a mAP_0.5 of 0.361 or 36.1% and a mAP_0.5:0.95 of 0.260 or 26.0%.

5.7.4 YOLOv5s, YOLOv5m, YOLOv5l

The YOLOv5 architecture comes in different size models, so far this paper has near exclusively focused on the YOLOv5s but it was felt that testing of other size models should be carried out for a comparison. A collection of graphs can be seen below

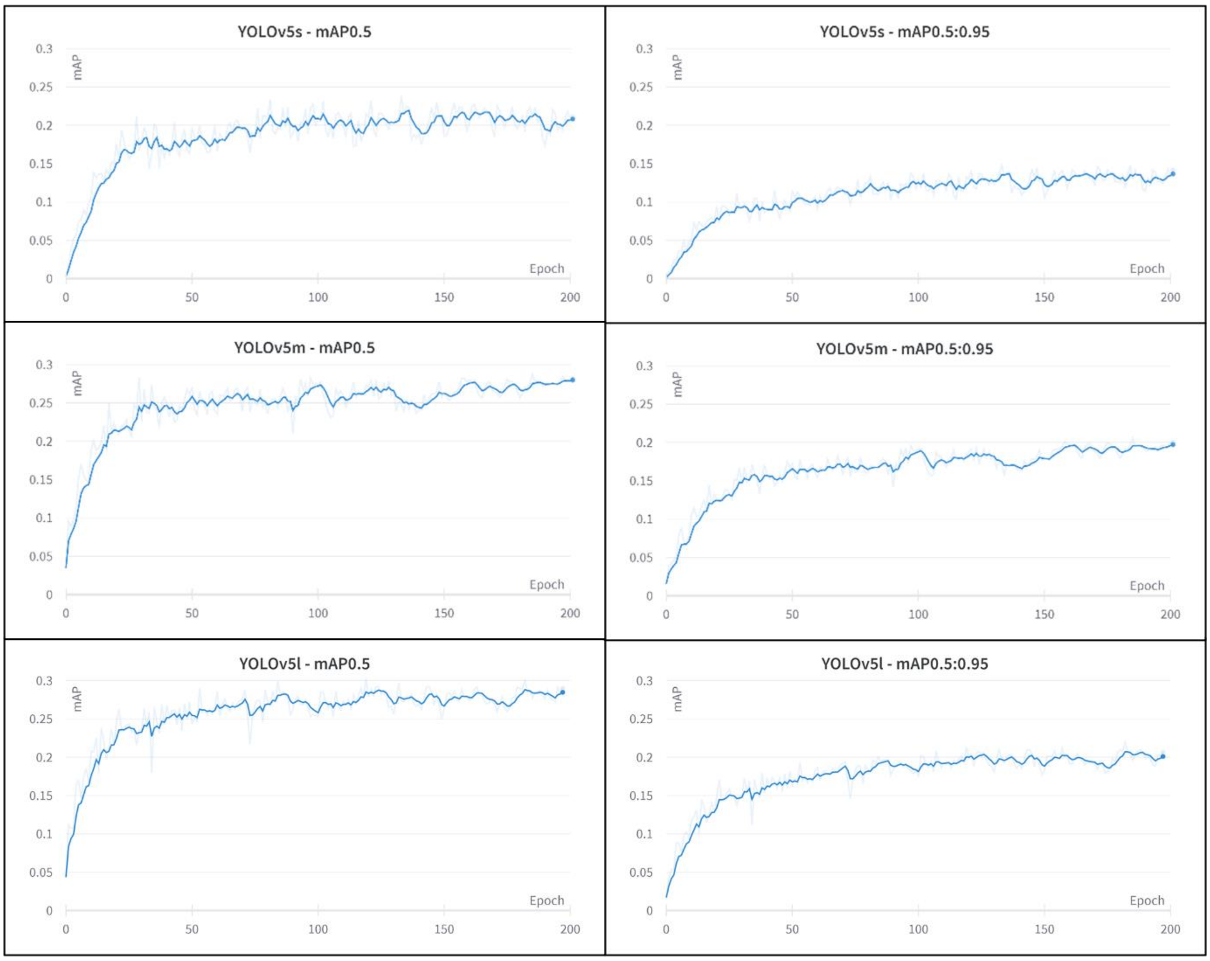


Figure 29 - A comparison of different size YOLOv5 models on the same dataset.

demonstrating the precision differences between the different models when trained on the same variation of the new dataset for 200 training cycles.

The YOLOv5s model as previously mentioned is made up of 270 layers, the YOLOv5m and YOLOv5l are made up from 369 and 468 layers respectively. This therefore means a large increase in weight parameters with the models containing 7.04 million, 20.9 million and 46.2 million from small to large. The graphs show that the medium and large models to perform better but they all converge to their maximum at around the same pace. The medium and large models took significantly longer time to train and when testing inference on a webcam the medium and large models were unusable with a CPU.

6 Discussion

This project has highlighted the challenges that must be confronted if object detection is going to be effectively utilised to solve a problem as complex as litter detection. As the results have shown, what success comes down to is the quality of the dataset. As expressed in section 3 management of the dataset took a lot of time. With a poorly annotated dataset the examples are working against the aim of the detector instead of with it. The realisation that a new dataset should be created to rule out a significant variable was good for results. Combining suitable images from both TACO datasets as well as self-collected images took a lot of time out of the project but ultimately proved that accurately detect something as variable as litter many more examples are required.

6.1.1 Cloud Computing

The use of cloud computing software has positives and negatives for the project. Without the ownership of a GPU it is invaluable to be able to quickly make small changes when training different versions compared to loading jobs to a supercomputer. The cloud computing platforms are build around having a very usable interface which lends itself well to those new to research in this area. That being said there was a lot of frustration with timing out when away from the internet window meaning that the training run would have to be restarted. There is a lot of supporting literature online that aids in the running of scripts on Google Colab which made it suitable for this project. Unfortunately the GPUs utilised with Google Colab can not be used for inference with a webcam, when meant that any tests with a webcam had to be done using the compute power of a CPU.

Roboflow was invaluable to the project after the initial problems with conversion. It very easily enables a user to make changes to a dataset and prototype with different augmentations and image settings. This project in total completed over 100 dataset variations when looking for improvements or seeing how specific object classes performed. To do this with a dataset stored locally this would very difficult and time consuming to do.

6.1.2 YOLO

YOLO stood out over other architectures when compared due to its quick inference speeds and accessibility. It has become extremely popular within the object detection space and due to the massive amounts of supporting literature seems to be the first step when researchers are entering the field and making models. A negative of YOLO however was due to the packaged-up nature when running the model doesn't initially allow for much opportunity to debug the system when it is performing poorly. YOLO accepts bounding box annotations which may not be best suited for this project. When annotating longer images such a plastic straw there is a lot of background in the bounding box compared to the object itself. This potentially could have been one of the reasons why the model struggled to differentiate between the object and the background. A different model with annotations in the polygon format would probably have had an increased accuracy, but at the cost of inference speed and the desire for the detector to be in real-time.

6.1.3 Future Work

There is a lot of ground to make with this topic and object detection as a whole. The problem of litter detection (and collection) lends itself well to an autonomous robot and with GPUs such as the NVIDIA Jetson boards give the compute required to run heavier networks than those tested in this project. This topic is still throttled by the lack of usable data. As proven by the unverified dataset outsourcing data annotation comes with its risks and the requirements to annotate a dataset of sufficient size is extremely difficult.

7 References

[1] Mark Rowe “Britain’s Growling Litter Problem; why is it so bad and how to take action.” countryfile.com. https://rb.gy/3jiptd (accessed Apr. 09, 2022)

[2] Keep Britain Tidy. “The Local Environmental Quality Survey of England 2017/18”. Keepbritaintidy.org. https://rb.gy/lfgt51 (accessed Apr. 09, 2022).

[3] UK Parliament. “Plastic Bottle Waste in the UK”. parliament.uk. https://rb.gy/elygs6 (accessed Apr. 09, 2022).

[4] R. Tehan et al., “A social experiment to understand how the presence of certain littered items influences rates of littering”. Journ. of Litter and Envir. Qual, vol 1. no.1, pp 5-15, June 2017.

[5] Canal and River Trust. “Plastic and litter in our canals”. canalrivertrust.org. https://canalrivertrust.org.uk/news-and-views/features/plastic-and-litter-in-our-canals (accessed Apr. 09, 2022).

[6] Department for Environmental Food and Rural Affairs. “Litter and littering in England 2018 to 2019.” gov.uk. https://rb.gy/yqvduf (accessed Apr. 09, 2022)

[7] Z. Zou, Z. Shi, Y. Gou and J. Ye, “Object Detection in 20 Years: A Survey,” *arXiv preprint, arXiv:1905.05055v2,* May. 2019.

[8] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proc. Of 2001 IEEE Comp. Soc. Conf on CVPR.* Kauai, HI, USA, pp. I-511 – I-518, doi: 10.1109/CVPR.2001.990517.

[9] P. Viola and M. Jones, “Robust Real-Time Face detection”, *Int. Journ. of Computer Vision.* vol.57, iss.2, pp.137-154, May 2004, doi: 10.1023/B:VISI.0000013087.49260.fb.

[10] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection” in *Proc. Of 2005 IEEE Comp. Soc. Conf on CVPR.* San Diego, CA, USA, pp.886-893, doi: 10.1109/CVPR.2005.177.

[11] Krizhenvksy. A, Sutskever. I and Hinton. G, “ImageNet classification with deep convolutional neural networks” *Comms. Of the ACM,* vol.60, iss.6, pp.84-90, 2017.

[12] Girshick. R, Donahue. J, Darrel. T and Malik. J, “Rich Feature Hierarchies for accurate object detection and semantic segmentation,” *arXiv preprint, arXiv:*1311.2524v5*,* Oct. 2014.

[13] Kelleher. J, *Deep Learning,* Cambridge, MA, USA: The MIT Press, 2019.

[14] Y. LeCun. “The MNIST DATABASE of handwritten digits”. www.yann.lecun.com. http://yann.lecun.com/exdb/mnist/ (accessed Apr. 15, 2022).

[15] Liangqu. L, Xingming. Z, *Beginning Deep Learning with TensorFlow: Work with Keras, MNIST, datasets and Advanced Neural Networks,* Berkeley, CA, Apress L.P, 2022. https://doi.org/10.1007/978-1-4842-7915-1

[16] Rodenblatt. F, “The Perception: A Probabilistic model for the information storage and organization within the brain”, *Psychological Review,* vol.65, no.6, pp.386-408, Apr. 1958, doi: 10.1037/h0042519

[17] Stanford CS. “CS231n Convolutional Neural Networks for Visual Recognition” cs231b.github.io. https://cs231n.github.io/convolutional-networks/ (accessed Apr. 22, 2022)

[18] Deep AI. “How does a Perceptron work?” deepai.org. https://deepai.org/machine-learning-glossary-and-terms/perceptron (accessed Apr. 13, 2022)

[19] K. Melcher. “A Friendly Introduction to [Deep] Neural Networks” Knime.com https://rb.gy/yh9mrd (accessed Apr. 13, 2022)

[20] A. Dertat. “Applied Deep Learning – Part 4: Convolutional Neural Networks” towardsdatascience.com. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 (accessed Apr. 20, 2022)

[21] D. Cornelisse. “An intuitive guide to Convolutional Neural Networks” www.freecodecamp.org. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/ (accessed Apr. 20, 2022)

[22] Kotikalapudi, Raghavendra et al. “keras-vis” github.io. https://raghakot.github.io/keras-vis/ (accessed Apr. 20, 2022)

[23] Valentini-Botinhao. Cassia, *“Noisy speech database for training speed enhancement algorithms and TTS models”,* Edinburgh, University of Edinburgh. Centre for Speech Technology Research (CSTR). 2016. [sound]. https://datashare.ed.ac.uk/handle/10283/2791 (accessed on 15/04/2022)

[24] Almeida. T.A, *“SMS Spam Collection Dataset”,* Sao Paulo, Brazil, Federal University of Sao Carlos (UFSCar). 2012. [text]. https://archive.ics.uci.edu/ml/datasets/sms+spam+collection (accessed on 15/04/2022)

[25] Kelleher. J and Tierney. B, *Data Science,* Cambridge, MA, USA: The MIT Press, 2018.

[26] J. Solawetz. “Train, Validation, Test Split for Machine Learning ” roboflow.com https://blog.roboflow.com/train-test-split/ (accessed Apr. 27, 2022)

[27] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[28] A. Krizhensky. “The CIFAR-10 Dataset”. www.cs.toronto.edu. http://www.cs.toronto.edu/~kriz/index.html (accessed Apr. 15, 2022).

[29] Lin. T-Y et. al, “Microsoft COCO: Common Objects in Context,” *arXiv preprint arXiv:* 405.0312v3*,* Feb. 2015.

[30] Google. “Open Images Dataset V6 + Extensions”. google.com. https://storage.googleapis.com/openimages/web/index.html (accessed Apr.15 2022)

[31] P. Kaur. “Convolution Neural Networks (CNN) for CIFAR-10 Dataset”. parneetk.github.io http://parneetk.github.io/blog/cnn-cifar10/ (accessed Apr. 15, 2022).

[32] A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollár, "Panoptic Segmentation," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9396-9405, doi: 10.1109/CVPR.2019.00963.

[33] S. Xie and Z. Tu, "Holistically-Nested Edge Detection," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1395-1403, doi: 10.1109/ICCV.2015.164.

[34] M. Fulton, J. Hong, M. J. Islam and J. Sattar, "Robotic Detection of Marine Litter Using Deep Visual Detection Models," *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 5752-5758, doi: 10.1109/ICRA.2019.8793975.

[35] Wolf. M et. al, “Machine learning for aquatic litter detection, classification and quantification (APLASTIC-Q)”, *Environ. Res. Lett.* vol. 15. no.11. Nov. 2020. doi: 10.1088/1748-9326/abbd01

[36] M. Fulton, J. Hong, M. J. Islam and J. Sattar, "Robotic Detection of Marine Litter Using Deep Visual Detection Models," *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 5752-5758, doi: 10.1109/ICRA.2019.8793975.

[37] Tata et al, “A Robotic Approach towards Quantifying Epipelagic Bound Plastic Using Deep Visual Models,” *arXiv preprint, arXiv* :2105.01882v4*,* Oct. 2021.

[38] Van Lieshout et al, “Automated River Plastic Monitoring Using Deep Learning and Cameras”, *Earth and Space Science,* vol.7, no.8, Jul. 2020. doi: 10.1029/2019EA000960.

[39] A. Chung, D. Y. Kim, E. Kwok, M. Ryan, E. Tan and R. Gamadia, "Cloud Computed Machine Learning Based Real-Time Litter Detection using Micro-UAV Surveillance," *2018 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2018, pp. 1-4, doi: 10.1109/URTC45901.2018.9244800.

[40] G. Thung, M. Yang, “Trashnet” github.com. http://github.com/garythung/trashnet (accessed Apr.17, 2022)

[41] P. F. Proença, “TACO” tacodataset.org. http://tacodataset.org/ (accessed Apr. 17, 2022)

[42] P. F. Proença and P. Simões, “TACO: Trash Annotations in Context for Litter Detection,” *arXiv preprint,* arXiv:2003.06975v2*,* Mar. 2020.

[43] M. Fulton, J. Hong. S. Junaed. “Trash-ICRA19: A Bounding Box Labelled Dataset of Underwater Trash” conservancy.umn.edu. https://conservancy.umn.edu/handle/11299/214366 (accessed Apr. 17, 2022)

[44] M. Kraft, M. Piechocki, B. Ptak, K. Walas, “Autonomous, Onboard Vision-Based Trash and Litter Detection in Low Altitude Aerial Images Collected by an Unmanned Aerial Vehicle”, *Remote Sensing,* vol.13, no.5, pp.1-17, Mar. 2021, doi: 10.3390/rs13050965

[45] M. Kraft, M. Piechocki, B. Ptak, K. Walas. “UAVVaste dataset” github.com. https://github.com/UAVVaste (accessed Apr.17, 2022)

[46] M. Fulton, J. Hong. S. Junaed. “TrashCan 1.0 An Instance-Segmentation Labeled Dataset of Trash Observations” conservancy.umn.edu. https://conservancy.umn.edu/handle/11299/214865 (accessed Apr. 17, 2022)

[47] M. Fulton, J. Hong. S. Junaed, “TrashCan: A Semantically-Segmented Dataset towards Visual Detection of Marine Debris,” *arXiv preprint,* arXiv:2007.08097v1*,* Jul. 2020.

[48] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.

[49] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[50] J. Redmon, A. Farhadi, “YOLO9000: Better, Faster, Stronger,” *arXiv preprint,* arXiv: 1612.08242v1*,* Dec. 2017.

[51] Hui. J. “Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3” medium.com. https://rb.gy/f6rkdx (accessed Apr.17, 2022)

[52] J. Redmon, A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv preprint,* arXiv: 1804.02767v1*,* Apr. 2018.

[53] Bochkovskiy. A. et al, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv preprint,* arXiv: 2004.10934v1*,* Apr. 2020.

[54] Ultralytics. “YOLOv5” github.com. https://github.com/ultralytics/yolov5 (accessed Apr.15, 2022)

[55] Redmon, J. “Darknet: Open Source Neural Networks in C” pjreddie.com. https://pjreddie.com/darknet/ (accessed Apr.29, 2022)

[56] Paszke. A, et al, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *arXiv preprint,* arXiv: 1912.01703v1*,* Dec. 2019.

[57] P. F. Proença, “TACO” github.com. https://github.com/pedropro/TACO (accessed Apr. 29, 2022)

[58] Supercomputing Wales, “Supercomputing Wales Portal” supercomputing.wales. https://portal.supercomputing.wales/ (accessed Apr.19, 2022)

[59] Google Colab, “Welcome to Colaboratory” colab.research.google.com. https://colab.research.google.com/notebooks/intro.ipynb (accessed Apr. 19, 2022)

[60] Roboflow, “Roboflow” roboflow.com/ https://roboflow.com/ (accessed Apr.19 2022)

[61] S. Yohanandan, “mAP (mean Average Precision) might confuse you!”, towardsdatascience.com. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2
(accessed on Apr. 20, 2022)

[62] L. Biewald, “Experiment Tracking with Weights and Biases,” Weights & Biases. [Online]. Available: http://wandb.com/ (accessed: Apr. 20, 2022).

[63] University of Saskatchewan, “Global Wheat Detection” kaggle.com. https://www.kaggle.com/competitions/global-wheat-detection/overview/evaluation (accessed Apr.19, 2022)

8 Appendices

8.1 Appendix A

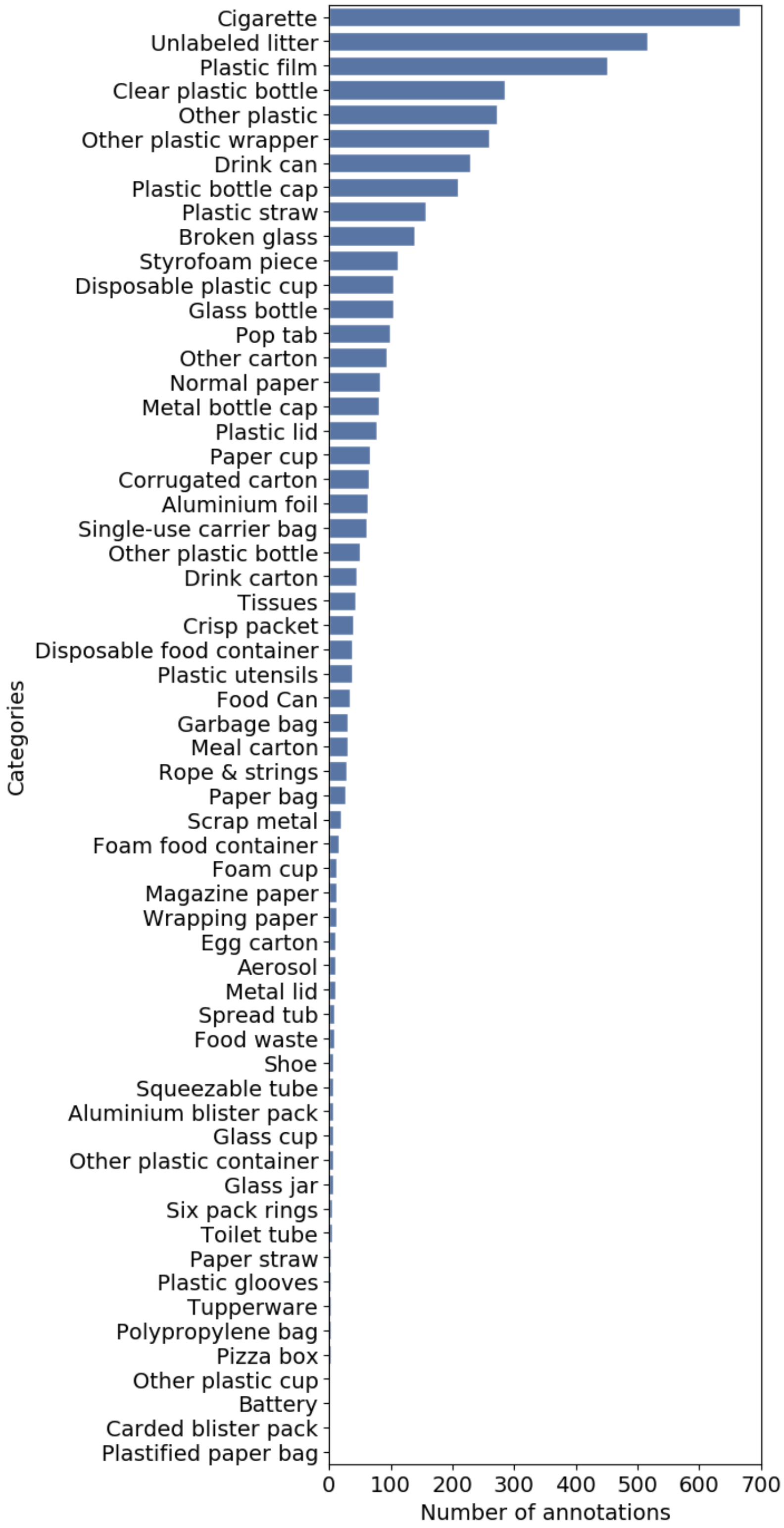
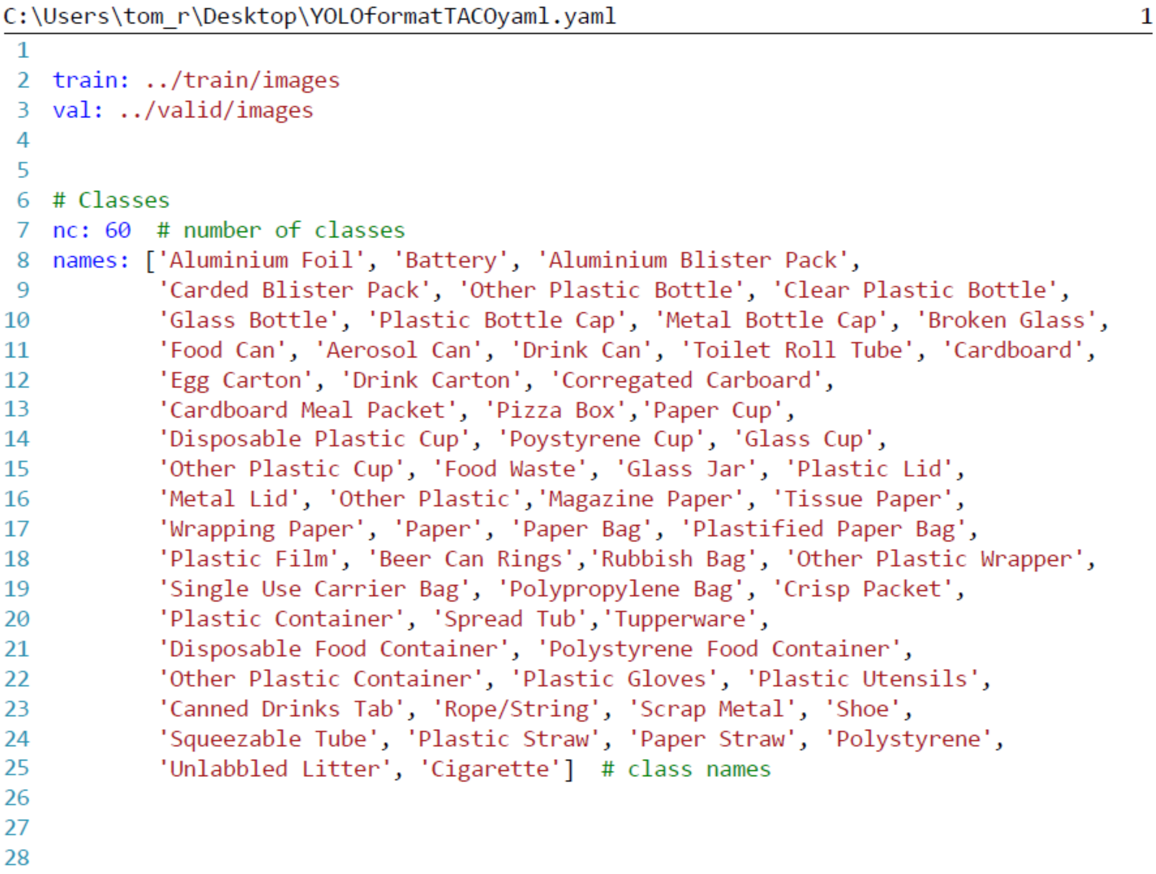


FIGURE 2 - A breakdown of the number of annotations per category in stock TACO dataset. Replicated from [36].

8.2 Appendix B

```
"info": {
  "description": "COCO 2017 Dataset",
  "url": "http://cocodataset.org",
  "version": "1.0",
  "year": 2017,
  "contributor": "COCO Consortium",
  "date_created": "2017/09/01"
}
"licenses": [
  {
    "url": "http://creativecommons.org/licenses/by-nc-sa/2.0/",
    "id": 1,
    "name": "Attribution-NonCommercial-ShareAlike License"
  },
  {
    "url": "http://creativecommons.org/licenses/by-nc/2.0/",
    "id": 2,
    "name": "Attribution-NonCommercial License"
  },
  ...
]
"images": [
  {
    "license": 4,
    "file_name": "000000397133.jpg",
    "coco_url": "http://images.cocodataset.org/val2017/000000397133.jpg",
    "height": 427,
    "width": 640,
    "date_captured": "2013-11-14 17:02:52",
    "flickr_url": "http://farm7.staticflickr.com/6116/6255196340_da26cf2c9e_z.jpg",
    "id": 397133
  },
  {
    "license": 1,
    "file_name": "000000037777.jpg",
    "coco_url": "http://images.cocodataset.org/val2017/000000037777.jpg",
    "height": 230,
    "width": 352,
    "date_captured": "2013-11-14 20:55:31",
    "flickr_url": "http://farm9.staticflickr.com/8429/7839199426_f6d48aa585_z.jpg",
    "id": 37777
  },
  ...
]
"categories": [
  {"supercategory": "person","id": 1,"name": "person"},
  {"supercategory": "vehicle","id": 2,"name": "bicycle"},
  {"supercategory": "vehicle","id": 3,"name": "car"},
  {"supercategory": "vehicle","id": 4,"name": "motorcycle"},
  {"supercategory": "vehicle","id": 5,"name": "airplane"},
  ...
  {"supercategory": "indoor","id": 89,"name": "hair drier"},
  {"supercategory": "indoor","id": 90,"name": "toothbrush"}
]
"annotations": [
  {
    "segmentation": [[510.66,423.01,511.72,420.03,...,510.45,423.01]],
    "area": 702.1057499999998,
    "iscrowd": 0,
    "image_id": 289343,
    "bbox": [473.07,395.93,38.65,28.67],
    "category_id": 18,
    "id": 1768
  },
  ...
  {
    "segmentation": {
      "counts": [179,27,392,41,...,55,20],
      "size": [426,640]
    },
    "area": 220834,
    "iscrowd": 1,
    "image_id": 250282,
    "bbox": [0,34,639,388],
    "category_id": 1,
    "id": 900100250282
  }
]
```

8.3 Appendix C

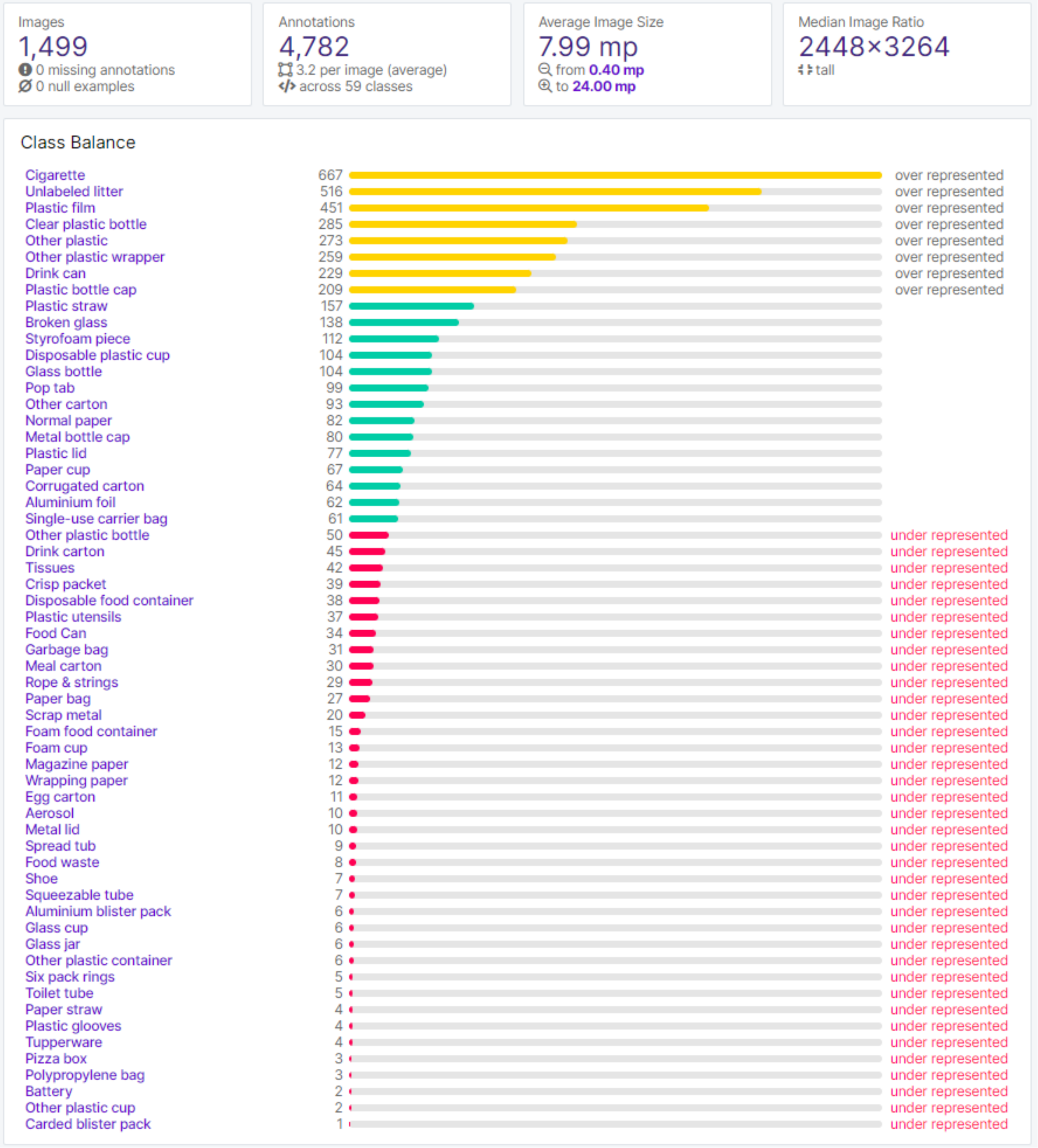


```
C:\Users\tom_r\Desktop\YOLOformatTACOyaml.yaml                                  1

train: ../train/images
val: ../valid/images


# Classes
nc: 60  # number of classes
names: ['Aluminium Foil', 'Battery', 'Aluminium Blister Pack',
        'Carded Blister Pack', 'Other Plastic Bottle', 'Clear Plastic Bottle',
        'Glass Bottle', 'Plastic Bottle Cap', 'Metal Bottle Cap', 'Broken Glass',
        'Food Can', 'Aerosol Can', 'Drink Can', 'Toilet Roll Tube', 'Cardboard',
        'Egg Carton', 'Drink Carton', 'Corregated Carboard',
        'Cardboard Meal Packet', 'Pizza Box','Paper Cup',
        'Disposable Plastic Cup', 'Poystyrene Cup', 'Glass Cup',
        'Other Plastic Cup', 'Food Waste', 'Glass Jar', 'Plastic Lid',
        'Metal Lid', 'Other Plastic','Magazine Paper', 'Tissue Paper',
        'Wrapping Paper', 'Paper', 'Paper Bag', 'Plastified Paper Bag',
        'Plastic Film', 'Beer Can Rings','Rubbish Bag', 'Other Plastic Wrapper',
        'Single Use Carrier Bag', 'Polypropylene Bag', 'Crisp Packet',
        'Plastic Container', 'Spread Tub','Tupperware',
        'Disposable Food Container', 'Polystyrene Food Container',
        'Other Plastic Container', 'Plastic Gloves', 'Plastic Utensils',
        'Canned Drinks Tab', 'Rope/String', 'Scrap Metal', 'Shoe',
        'Squeezable Tube', 'Plastic Straw', 'Paper Straw', 'Polystyrene',
        'Unlabbled Litter', 'Cigarette']  # class names
```

8.4 Appendix D

```
# Convert Coco JSON Annotations to YOLO TXT Files

import logging
logging.getLogger().setLevel(logging.CRITICAL)
!pip install pylabel > /dev/null

from pylabel import importer

!git clone https://github.com/pedropro/TACO

!cd TACO>detector && python3 download.py --dataset_path
./data/annotations.json

"""## Import coco annotations
Import annotations from the coco dataset, which are in coco json format.
"""

import os
import zipfile
import cv2

#Specify path to the coco.json file

path_to_annotations = "/content/TACO/data/annotations.json"
#Specify the path to the images (if they are in a different folder than the
annotations)
path_to_images = "/content/TACO/data/official"

#Import the dataset into the pylable schema
dataset = importer.ImportCoco(path_to_annotations,
path_to_images=path_to_images, name="Litter_COCO")
dataset.df.head(10)

# Analyze annotations

print(f"Number of images: {dataset.analyze.num_images}")
print(f"Number of classes: {dataset.analyze.num_classes}")
print(f"Classes:{dataset.analyze.classes}")
print(f"Class counts:\n{dataset.analyze.class_counts}")
print(f"Path to annotations:\n{dataset.path_to_annotations}")

"""# Export to Yolo v5
Yolo creates one text file for each image in the dataset.
"""

dataset.path_to_annotations = "/content/TACO/data"
dataset.export.ExportToYoloV5(output_path='/content/TACO/data',
yaml_file='dataset.yaml',copy_images=False, use_splits=False,
cat_id_index=None)

from google.colab import drive, files
drive.mount('/content/drive')

!zip -r /content/TACO/data/official.zip /content/TACO/data/official
```

8.5 Appendix E



Images
1,499
0 missing annotations
0 null examples

Annotations
4,782
3.2 per image (average)
across 59 classes

Average Image Size
7.99 mp
from 0.40 mp
to 24.00 mp

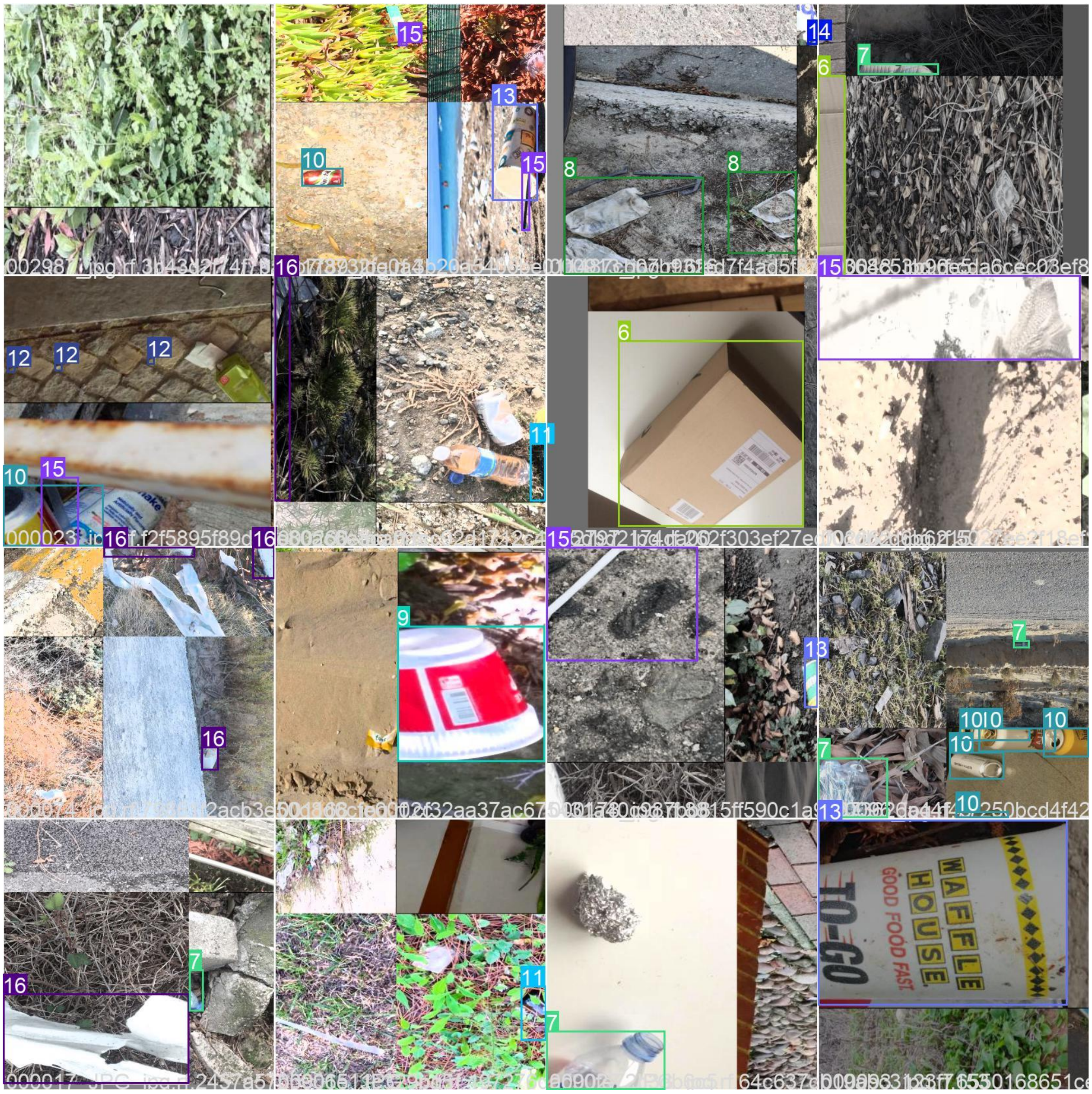
Median Image Ratio
2448×3264
tall

Class Balance

Cigarette	667	over represented
Unlabeled litter	516	over represented
Plastic film	451	over represented
Clear plastic bottle	285	over represented
Other plastic	273	over represented
Other plastic wrapper	259	over represented
Drink can	229	over represented
Plastic bottle cap	209	over represented
Plastic straw	157	
Broken glass	138	
Styrofoam piece	112	
Disposable plastic cup	104	
Glass bottle	104	
Pop tab	99	
Other carton	93	
Normal paper	82	
Metal bottle cap	80	
Plastic lid	77	
Paper cup	67	
Corrugated carton	64	
Aluminium foil	62	
Single-use carrier bag	61	
Other plastic bottle	50	under represented
Drink carton	45	under represented
Tissues	42	under represented
Crisp packet	39	under represented
Disposable food container	38	under represented
Plastic utensils	37	under represented
Food Can	34	under represented
Garbage bag	31	under represented
Meal carton	30	under represented
Rope & strings	29	under represented
Paper bag	27	under represented
Scrap metal	20	under represented
Foam food container	15	under represented
Foam cup	13	under represented
Magazine paper	12	under represented
Wrapping paper	12	under represented
Egg carton	11	under represented
Aerosol	10	under represented
Metal lid	10	under represented
Spread tub	9	under represented
Food waste	8	under represented
Shoe	7	under represented
Squeezable tube	7	under represented
Aluminium blister pack	6	under represented
Glass cup	6	under represented
Glass jar	6	under represented
Other plastic container	6	under represented
Six pack rings	5	under represented
Toilet tube	5	under represented
Paper straw	4	under represented
Plastic glooves	4	under represented
Tupperware	4	under represented
Pizza box	3	under represented
Polypropylene bag	3	under represented
Battery	2	under represented
Other plastic cup	2	under represented
Carded blister pack	1	under represented

Figure 3 - The "health check" on the TACO dataset showing the spread of annotations

8.6 Appendix F



8.7 Appendix F

Record of project meetings

Date: 08/10/21	**Time: 16:00**	**Location/Platform: Microsoft Teams**
Agenda / discussion items	**We discussed what programming language would be more appropriate for this deep learning problem. We also talked about some possible objectives for the project such as comparing different neural network architectures.**	
Action(s) by next meeting	• **Determine whether I should use the supercomputer** • **Determine whether a pre-trained network should be used** • **Investigate some suitable datasets for training**	

Date: 19/10/21	**Time: 10:00**	**Location/Platform: Microsoft Teams**
Agenda / discussion items	**The project learning objectives were discussed in preparation for the coursework assignment. Libraries like OpenCV will be useful to this project. Should this project eventually be mobile deployable? Deep learning toolbox on MATLAB is good for getting a clearer understanding of deep learning.**	
Action(s) by next meeting	• **More research into datasets** • **Decide on project learning objectives** • **Research PyTorch and OpenCV**	

Date: 26/10/21	**Time: 10:00**	**Location/Platform: Microsoft Teams**
Agenda / discussion items	**We talked about the computation requirements of training a model, the use of the Hawk supercomputer will be beneficial. The supercomputer will require some use of Linux. The project could be purely training or trying to finetune a network**	
Action(s) by next meeting	• **Look on GitHub and Kaggle for similar projects** • **Explain to Yulia how they went about their projects in the next meeting** • **Decide on the dataset**	

Date: 02/11/21	**Time: 10:00**	**Location/Platform: Microsoft Teams**
Agenda / discussion items	**The supercomputer is coded on the command line, i.e. does not have an IDE. Continue working with the open-source code and trying to get it to work without success so far. I could do something similar to the TACO project and opensource dataset but compare the accuracy with another architecture.**	
Action(s) by next meeting	• **I need to ensure that the correct libraries have been installed on the supercomputer**	

Date: 09/11/21	Time: 10:00	Location/Platform: Microsoft Teams
Agenda / discussion items	I managed to train an object detection programme to recognise screwdrivers. In the process of doing this it has cleared up a lot of questions regarding the process and stages involved in object detection and instance segmentation. You can more easily use a pretrained network where you use pretrained weights and adapt the last layers of the neural network to out cause.	
Action(s) by next meeting	• I need to start the application for the supercomputer • Which architecture?	

Date: 16/11/21	Time: 10:00	Location/Platform:
Agenda / discussion items	Feedback from the project brief was that the literature review requires more attention. We discussed that a good literature review is a discussion not a summary. We also talked about the upcoming progress review presentation.	
Action(s) by next meeting	• For next week I need to work through the tutorial written by the author of the commonly used Mask R-CNN	

Date: 23/11/21	Time: 10:00	Location/Platform: Microsoft Teams
Agenda / discussion items	We mentioned some of the other neural network architectures that are used in systems such as fast R-CNN, Faster R-CNN and YOLO. At the moment just be concerned about a high-level explanation of how it works.	
Action(s) by next meeting	• Read through the guide in the project handbook reference the presentation and keep familiarising with the TACO code	

Date: 30/11/21	Time: 10:00	Location/Platform: Microsoft Teams
Agenda / discussion items	We discussed in more detail what is required in the progress review presentation. We discussed some points on the high-level overview of the project and points to include in the presentation. We also talked about some points regarding the GPU allowances on Google Colab vs. the supercomputer.	
Action(s) by next meeting	• Start on the presentation slides and have first draft ready for discussion next week.	

Date: 07/11/21	Time: 10:00	Location/Platform: Microsoft Teams
Agenda / discussion items	We talked through the first draft of my project review presentation	
Action(s) by next meeting	• Amend the presentation where specified, add slides to show that there was a thought process when choosing the TACO dataset also to include a brief intro into neural networks and deep learning	

Date: 16/11/21	Time: 10:00	Location/Platform: Microsoft Teams
Agenda / discussion items	Project review presentation	
Action(s) by next meeting	• Keep working on code and researching the topic as a whole	

Date:14/02/22	Time:14:00	Location/Platform: Microsoft Teams
Agenda / discussion items	The importance of keeping good notes when going through training and taking a lot of screenshots. The time commitment for the project is not 45+ hours a week. Start the report while going through training runs.	
Action(s) by next meeting	• Try to get something working • Look into suitable architectures • Demonstration using hawk for next week	

Date:21/02/22	Time:13:00	Location/Platform: Microsoft Teams
Agenda / discussion items	We discussed some of challenges with using hawk supercomputer. How I had gone though some of the tutorials on how to us various aspects of it. I had learnt a lot about using the command line to clone repositories and download datasets	
Action(s) by next meeting	• Get more experience with hawk and try to run a script • Try to get the dataset converted	

Date:28/02/22	Time:13:00	Location/Platform: Microsoft Teams
Agenda / discussion items	The litter dataset has too many classes. When trying to do my own annotations its it hard to compact them into better classes that are not too variable. Should I merge items together such as all plastic bottles? I need to make sure I can justify all decision that I make in the project for the report. The score with the bounding box is just a probability/confidence score.	
Action(s) by next meeting	• Try to get the dataset converted • Get rid of any classes that cause too much confusion	

Date:07/03/2022	Time:13:00	Location/Platform: Microsoft Teams
Agenda / discussion items	The upcoming progress review. The network would be biased to over represented classes, i.e cigarettes. The model will overfit when there is not enough data and will not generalise well.	
Action(s) by next meeting	• Test with variations of the dataset and try to use classes with a similar number of annotations	

Date:14/03/2022	Time:13:00	Location/Platform: Microsoft Teams
Agenda / discussion items	When should we start writing the report? Make sure I use augmentations such as flip and rotate. We discussed the different parts of the dataset, ie test, train and val. We also discussed confusion matrixes	
Action(s) by next meeting	• Try to eliminate problems with the dataset such as similar looking classes and poor annotations.	

Date:21/03/2022	Time:13:00	Location/Platform: Microsoft Teams
Agenda / discussion items	What is an acceptable accuracy for a model? Near to 50%. Ensure that I note down low accuracy. The last fully connected layers are classifiers. Make sure I document everything.	

Action(s) by next meeting	• **Think about the structure of my report. Write a few sentences about what will be written about in each of the main sections.**

Date:25/04/2022	**Time:13:00**	**Location/Platform: Microsoft Teams**
Agenda / discussion items	**Make sure that in the report you highlight all of the steps that I went through in the process of the project. Ensure that captions give a description of the image.**	
Action(s) by next meeting	**N/A**	