Litter Coastal Image Analysis in Scotland

Automatically Litter Detection from SCRAPbook Dataset using Faster R-CNN

EE997 – Individual MSc Final Project

Presented by

Chavinpat Naimee (201976778)

The study was supervised by

Dr Marc Roper

MSc in Machine Learning and Deep Learning

Department for Electronic and Electrical Engineering

21 August 2020

University of Strathclyde

Abstract

Marine debris has a severe impact on marine ecosystems that is important for human life. SCRAPbook project has been developed to conserve the ocean by using aerial images that were captured from a small aircraft and then identify litter on an individual image manually. However, it could take a long time and a high cost for the task because there are numerous numbers of images. Therefore, this project has been constructed and developed a model using machine learning and deep learning that enables automatically to detect the presence of litter in aerial pictures. The technique of computer vision called object detection can deal with this issue. This project then presents a deep learning-based automatic litter detection system that allows users to save up to a higher amount of time as they can use this system to detect aerial images. The experiment was divided into two parts. Firstly, it examines how the performance change if the number of labelled images and classes are increased by using Faster R-CNN inception v2 and 20000 training steps. The second experiment is to discover a high-performance model for our data that are trained using Faster R-CNN Inception Resnet v2 and Faster R-CNN Resnet 101 COCO. Finally, these models are tested on unseen data to investigate efficiency. The results are presented using the COCO matric. The results show that the mAP score would increase if the number of labelled training data is increased. Moreover, models that trained by Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2 outperformed the model that trained from Faster R-CNN Inception v2, including for testing data. As a result, Object detection model using deep learning for litter detection shows pretty good performance which can allow the user to detect litter automatically.

Keywords: coastal cleaning; litter detection; object detection; computer vision; deep learning; Faster R-CNN

Table of Contents

Table of Figures

Table of Tables

Chapter 1 Introduction

1.1. General Background

Litter has been deposited in every environment that most local governments and international organisations struggle to address this issue. It has many effects on biodiversity, especially in marine life, including physical, biological, and chemical impacts [1]. For example, marine creatures could misunderstand that plastic is their food, as shown in figure 1-1. This issue should be solved to avoid marine ecosystems from destruction.

Figure 1-1 Turtle eats a plastic bag [2].

Litter on the beach or in the ocean, called marine debris, is solid materials made by human that is discarded or disposed of on beaches, in drains lead to the sea, or into the ocean. Marine debris, such as plastic grocery bags, soda bottles or cans, cigarettes, fishing equipment and other manufactured materials, is found on beaches and in the ocean around the world. However, it does not contain naturally derived materials such as dead seaweed, carcasses, and shells. Most of all marine debris is composed of plastic that is the most dangerous marine debris because it takes 500-1000 years to decompose [3]. Ocean Conservancy's Trash Free Seas Alliance reports that 8.3 million tons of plastics are discarded in the ocean yearly [4]. The research of Condorferries [5] said that 70% of trash sinks, 15% floats in the sea, and 15% is on the beaches.

1.2. Motivation

The Scottish Coastal Rubbish Aerial Photography or SCRAPbook [6] is an innovative project to conserve the ocean organised by UK Civil Air Patrol (Sky Watch), Moray Firth Partnership and Marine Conservation Society. The purpose of this is to clean Scotland's coast using aerial

images taken from the light aircraft by mapping litter that appear in images. The image data from SCRAPbook with geographical positioning information can be used for mapping [5]. They captured useful photos when it is low tide conditions that litter on the foreshore will be noticed. Figure 1-2 illustrates the example of aerial imageries that contain litter. After they collect data, staffs or volunteers will analyse an image one by one to classify the intensity and type of the trash appear from the image. Moreover, they are also looking at how is it distributed, how much there is, and other useful information would be extracted. Finally, they put these data in a spreadsheet template. However, this work will task ages to finish manually.

Figure 1-2 Sample aerial images from SCRAPbook project [7].

In order to build the model doing the task instead of human, several challenges have been developed. Firstly, the images that obtained from SCRAPbook is quite low quality and blurred. Data is one of the most crucial factors when training the model, but data transformation or data augmentation techniques could help to solve this issue. Another challenge is the number of data/images is relatively small, which is one problem in deep learning. A small amount of data may provide a poor performance of the model. Nevertheless, transfer learning method, which discussed later, would help.

1.3. Objectives and Scope

The goal of this project is to build and develop a model using machine learning and deep learning that enables automatically to detect the presence of trash in aerial pictures. Firstly, this project will focus on a review of literature and method of relevant topics, i.e. marine debris issues, SCRAPbook project, computer vision and applications, including image classification, object detection, and image segmentation, and deep learning technique for computer vision. Besides, deep learning for object detection, the proper Convolutional Neural Network (CNN)

architectures for litter detection, and the comparison of performance in each CNN architectures will be researched, especially for Faster R-CNN [8] model that would be the leading architecture for this project. Then experiment using the proper technique with our data that are taken from SCRAPbook, and exploration of the improvements in model efficiency using innovative machine learning and deep learning approaches.

The scope of this task is to look at the state-of-the-art CNN architectures that can detect small objects in an image. Besides, investigating how the behaviour of the performance when the number of class and labelled training data are varied in a low-resolution image with a small amount of data. The production of models will be evaluated using a metric from COCO challenge [9] that was adopted from mean Average Precision (mAP) matric.

1.4. Limitations

The algorithm built in this report has only experimented with a single dataset. In the real system, the model was not implemented. The number of images that contains litter is quite small and low quality. The images in the dataset must be resized because of the resource constraint of RAM in Google Colaboratory. Also, the restriction of running time no more than 12 hours on Google Colab could confine the number of training step, and the run-time sometimes restarts by itself.

1.5. Research Structure

Hence to inform the reader, this report is organised as follows. Section 2 presents the literature reviews of relevant topics, and Section 3 describes the methodology of the research and related techniques, including the experiment that will be tested in this project. Then, Section 4 reports the results and then analyse, evaluate and compare the obtained results using each model. Section 5 is a summary of the project. Finally, the report ends with references and appendices.

Chapter 2 Literature Review

2.1. Marine Debris Issues

Litter has been deposited in every environment that most local governments and international organisations struggle to address this issue. Any artificial object that has been discarded, disposed, or abandoned into the ocean or coastal areas, called marine litter or marine debris [10]. Practically, any sort of items ever made by humans has been discovered that discarded somewhere on the beach; everything can display on the beach, even kitchen sink [11]. It also includes any items from the land abandoned to the sea by rivers or other drainages. Marine debris, such as plastic grocery bags, soda bottles or cans, cigarettes, fishing equipment and other manufactured materials, is found on beaches and in the ocean around the world. However, it does not contain naturally derived materials such as dead seaweed, carcasses, and shells.

It seems that the marine environment is far away from our life, but it is not. Marine ecosystems, which are the most extensive system in the world, is essential for human life [11]. Marine litter has many effects on biodiversity, especially in marine life, including physical, biological, and chemical impacts [1]. For example, marine creatures could misunderstand plastics are their food, or they might be trapped from a fishing net that left in the ocean.

Most of all marine debris is composed of plastic that is the most dangerous marine debris because it takes 500-1000 years to decompose [3]. Ocean Conservancy's Trash Free Seas Alliance reports that 8.3 million tons of plastics are discarded in the ocean yearly [4]. Ferries [5] shows that 70% of trash sinks, 15% floats in the sea, and 15% is on the beaches.

There are several ways to tackle this crisis. This must be solved at the root of the problem; meanwhile, litter already in the ocean or on coastal should be removed at the same time. Nowadays, nonetheless, there is an increase in the number of organisations that concern about marine debris. SCRAPbook [6] is one of the projects from Scotland that concerns about marine debris.

2.2. SCRAPbook Project Task

The Scottish Coastal Rubbish Aerial Photography or SCRAPbook [6] is a creative project organised by UK Civil Air Patrol (Sky Watch), Moray Firth Partnership and Marine Conservation Society. The purpose of this is to clean Scotland's coast using aerial images taken from the light aircraft by mapping litter that appear in images. SCRAPbook [6] said that using aerial imagery is

a more effective technique of surveying the widespread coastal area. The atmospheric picture resolution is high enough to identify an object greater than 10 centimetres [7]. Another benefit to this method is the capability to inspect a remote or inaccessible area of shoreline. After obtained images, they would be sent to staffs or volunteers who would categorise them according to the intensity and distribution of litter noticeable in each picture [7].

The primary outcome of the project was the interactive SCRAPbook map, showing the spatial distribution and severity of macro-scale coastal litter by colour-coded dots, with photos attached to dots with the largest concentration of litters [7]. The map will also be used to notify different litter types in further, more detailed, on-the-ground surveys [7]. However, presently, the photos captured by the aircraft are hand-held analyses to detect visible rubbish, which consumed a lot of time to spot them.

For the first time, the SCRAPbook dataset has provided a spatially accurate description of the distribution of coastal litter hotspots across Scotland. This can be used to understand better the mechanisms behind accumulation and distribution of coastal litter, and to inform national marine planning and policies [7].

2.3. Computer Vision and Applications

Computer Vision or CV is an area of research relating to the development of techniques to allow computers are able to “see”. This means it can interpret and analyses image and video content and extract valuable information [12]. Recently, the computer vision field has developed from using statistical methods to using machine learning and deep learning techniques.

Basically, CV tasks are methods of collecting data, processing, pattern recognition and computer graphics. Most of the job is related to the purpose of receiving information from input data or digital images and then extracting features from them [13]. The techniques used to solve CV problems depend on the context and complexity of the data being analysed [12]. CV primarily aims to extract features and build models from input data and using image processing to implement computational transformations of images, including edge detection, contrast, filtering, etc. [13].

There are many techniques in computer vision have been developed and presented for real-world application. One of the well-known methods in CV is object recognition, including image classification, image localisation, object detection and image segmentation [12].

2.3.1. Image Classification

Image classification is a classical technique that has been used until now. This method is known as the classification of only a single object in an image [12]. For instance, figure 2-1(a), an image of a cat as an input, the classifier will predict the picture as a class label (cat's class). A multi-label classifier has been built over the past few years that is capable of predicting more than one category in an image. Conventional image classification begins with input as an image and pass-through feature extractor, which is hand feature engineering, to generate a set of a number represents the object [12]. There are many famous feature extractor or feature descriptors such as Harr [14], HOG [15], and SIFT [16]. After that, the set of the number will be used to classification and output class will be predicted. Currently, state-of-the-art image classifications usually use feature extractors from training dataset using Convolutional Neural Network (CNN) architectures instead of handcraft feature engineering.

2.3.2. Object detection and Localisation

However, the problem is how can we know the location of each item in the image. Localisation can help to identify the location of items in an image that is the primary concept of object detection, see figure 2-1(b) [12].

Object detection allows us to predict both the location and the class for each object, see figure 2-1(c). In fact, the location of an object should be located before predicting the class. There are two types of object detectors, which are one-stage and two-stage detectors [17]. Both start with bounding boxes or anchor boxes that can identify the location of an object. A two-stage one has two processes: a region proposal network and a classification step [17]. The region proposal network will present the bounding box with a high likelihood around the object, called anchor box. Whilst the classification network will take anchor boxes and predict their class. The region proposal and classification networks are merged into a single stage called one-stage detector, which the model can predict both systems at the same time [17]. The advantage of a one-stage detector is faster than two-stage because they are merged. However, the separation of the two tasks will provide higher accuracy.

2.3.3. Image Segmentation

Image segmentation, figure 2-1(d), along with classification and object detection is one of the unique methods in computer vision. Image segmentation is to split an image into separate sections that usually correspond to a specific region, such that each pixel is assigned to its part [12]. It seems that segmentation in an object is easy for machines as well; in fact, it is

complicated because it needs to classify every pixel in an image. Traditional image segmentation used k-means clustering, edge detection, or thresholding to divide an image. Recently, it has been shown that deep learning or CNN techniques work well in segmentation task, with results approaching or surpassing a human level [18].

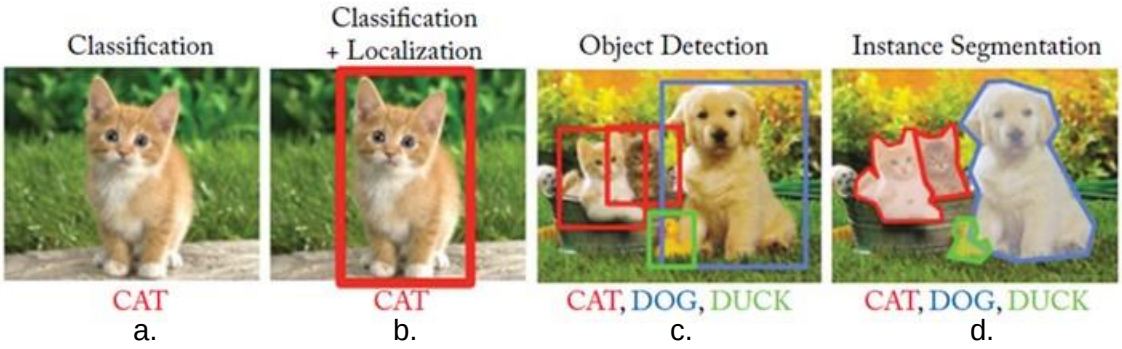


Figure 2-1 The difference of Image classification, localisation, object detection, and image segmentation [12].

2.4. Deep Learning

Deep learning is a subset of machine learning that allows machines to automatically learn and develop their tasks from experiences [19]. Deep learning methods have been used in many CV applications, such as image classification [18]. Deep learning approaches have four main types, which are supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning. Supervised learning methods require training data with manually labelled data or images that use to teach a model in order to make a prediction for unknown label [20]. CNNs method is one of the most popular techniques for supervised learning.

On the other hand, unsupervised and semi-supervised techniques would be used when a dataset lacks supervisors or labelled data. These models are able to solve problems with little or no information about how the results would be [20]. Unsupervised learning is used when there is no desired output, while semi-supervised is a hybrid combination of supervised and unsupervised learning that are using both labelled and unlabelled data to train the model.

Reinforcement learning is another effective learning algorithm. It is a learning by using the concept of interaction, agent, environment, and reward to achieve a task. This learning will learn to obtain maximum reward in a particular situation [21].

This project will focus on CNN models that have been developed and achieved high accuracy for object detection problems in many kinds of research.

2.4.1. Deep Learning for Object Detection

Object detection is known as a method of classifying and localising multiple objects in one image. In the past few years, pre-trained CNN was widely used to identify and locate a single object in an image then to slide a CNN window to search other items [22]. Figure 2-2 shows the image that contains multiple flowers in a different size [22]. This example shows that the image was divided into a 6x8 channels and a CNN window (the thick black rectangle) is 3x3 regions. The problem is a CNN window will detect the same object more than one times. For example, a CNN window starts at the top-left corner, and it will detect the top-left flower. It will identify the same flower again when the window was shifted one block to the right or even to the bottom. Besides, because flowers can vary in scale, other sizes of a CNN window will be considered, such as 4x4 regions. Even though it seems that this is a straightforward technique, but it still does not work well. However, unnecessary bounding boxes could be taken out by the method called non-max suppression [22]. The idea is to calculate the probability that a flower certainly appears in the image using a sigmoid activation function [22]. This simple method to detect objects seems to work well; however, it takes many training times to train the model.

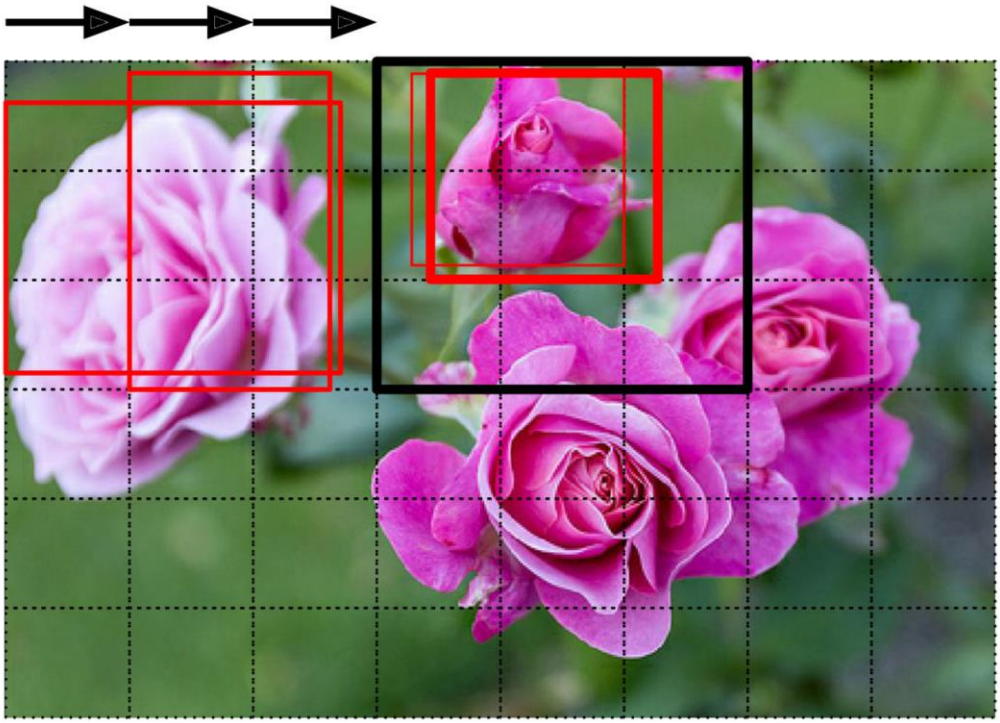


Figure 2-2 Multiple object detection by sliding a CNN window [22].

Nevertheless, fortunately, there are many deep CNN techniques have been developed to achieve successfully in speed and accuracy for object detection. Krizhexsky et al. [23] trained a large deep CNN using the ImageNet dataset, which contains more than a million high-resolution images, in LSVRC-2010 and LSVRC-2012 competitions [24] and they achieved top-5 rank in both contests. ImageNet is one of the most successful models for image classification [23]. In 2014, [25] proposed a new detection algorithm was trained on PASCAL VOC dataset that provided a high efficiency of mean average precision (mAP), which is an evaluation metric that will be discussed later. This approach took a technique of region proposals to locate and segment objects and used CNN pre-trained model in order to solve a small size of labelled data problem, called Regions with CNN features or R-CNN technique. Nonetheless, due to the multi-stage pipeline that contains feature extraction, fine-tuning network, training linear SVMs, and bounding-box regressors, large storage and time were needed for the first generation of R-CNN method. Other versions of R-CNN series will be presented in 2.2.2.

Another extremely fast and accurate architecture for object detection that is You Only Look Once (YOLO) algorithm [26], proposed by Joseph Redmon et al. The first YOLO generation was presented in 2015 and then has been developed to be YOLOv2 in 2016 and YOLOv3 in 2018. The benefit is YOLO can perform in a real-time situation on a video with maintaining acceptable accuracy.

2.4.1. R-CNN Architectures for Object Detection

One of the most widely used methods for object detection is the R-CNN family, developed by Ross Girshick et al. in 2014. The family contains R-CNN [25], Fast R-CNN [27], Faster R-CNN [8], and Mask R-CNN [28]. The difference between each R-CNN version is computational efficiency, improvement of speed, and increase performance.

R-CNN networks typically contain 4 things. Firstly, a region proposal method to create bounding boxes or position of potential objects in an image. Secondly, a feature extraction steps to generate features of those objects, regularly using a CNN model. Thirdly, a classification layer or softmax layer to predict the class of each object. Lastly, a regression layer to make the object bounding box coordinates more accurate.

Recent progress in object detection is motivated by the performance of the region proposal technique and region-based convolutional neural networks (R-CNNs). Even though R-CNN is computationally expensive as they were initially built-in, their costs were radically reduced thanks to the sharing of proposals [20]. The latest version has been developed to reach near

real-time object detection using very deep neural networks if the time spent on region proposal was ignored [8]. Thus, the problem of novel object detection of R-CNN series is the region proposal step.

However, state-of-the-art Region Proposal Networks (RPNs), which share convolutional layers with novel object detection, was presented, known as Faster R-CNN. [8] proposed the Faster R-CNN that was published in 2015 in ILSVRC and COCO competitions and succeeded in ImageNet detection, ImageNet localisation, COCO detection and COCO segmentation. [29] said that Faster R-CNN is not only a cost-effective solution for practical use but also increase the accuracy of object detection.

2.4.2. YOLO v3 Architecture for Object Detection

YOLO v3 is a deep CNN responsible for real-world object detection, including localising and classifying. It was trained on the Common Objects in Context or COCO dataset. Input data is split into SxS cell grid, where each grid cell is used for predicting the object whose centre falls within it. That grid cell predicts bounding boxes B along with probabilities of class C, which are the probabilities of the class in which the object is assumed to belong [30]. The overall output consists of the image centre coordinates (x and y), the box height and width (h and w), and the class confidence it was categorised into. Therefore, the total output of YOLO is SxSxBx5 [30]. Figure 2-3 shows the bounding box prediction by YOLO v3.

YOLO v3 is made up of 53 convolution layers, is known as Darknet-53, including 3x3 and 1x1 kernels, which are responsible for the extraction of the feature. This architecture is much faster than all similar networks while maintaining a high mean average precision (mAP) [31].

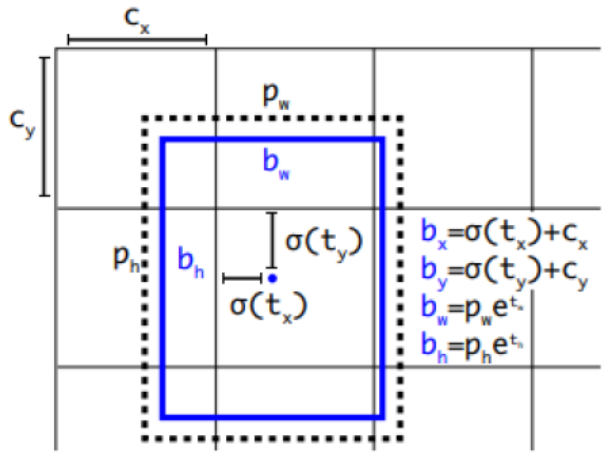


Figure 2-3 The bounding box prediction of YOLO v3 [30]

2.4.3. Comparison of Deep Learning Model for Object detection

Geron [22] suggested that there are many factors need to be considered in order to choose an appropriate architecture for our project such as speed, accuracy, complexity, training time, availability of the pre-trained model, etc. Dixit et al. [32] presented the comparison of the performance between YOLO and R-CNN for object detection, which accuracy and speed are primary factors. Figure 2-4 demonstrates that Faster R-CNN would provide the highest efficiency, while YOLO is the fastest algorithm. It can be seen that if real-time detection is more concerned than accuracy, YOLO is the best choice. However, Faster R-CNN is the best option if very high accuracy is necessary [32]. For example, an autonomous driving system requires a real-time detection on a video; thus, YOLO algorithm could be suitable. While litter detection, like our project, could use Faster R-CNN because speed does not need a concern much [20].

Figure 2-4 Comparison between speed and accuracy of R-CNN and YOLO [32]

2.5. Litter Detection using Faster R-CNN

The paper “Smart Street Litter Detection and Classification Based on Faster R-CNN and Edge Computing” [20] is research of using Faster R-CNN to detect litter on the street by using an automated method to control the road cleaning machine. The paper designed and developed a smart city application using the state-of-the-art of Internet of Thing (IoT), mobile edge computing, big data analysis, and machine learning method. The exciting part of this research is that they used Faster R-CNN to detect litter on the street, which can be adapted to our project. They have developed the model to detect, classify and analyse various kinds of litter on the road such as leaves, tree branches, plastics, bottles, etc.

This research used 12,346 of litter images for training deep learning detection approach and detected 11 categories of the litter with good efficiency. They experimented on two potential case studies. The first case study is training on a small dataset with 6 classes of litter, which are plastic bags, plastic cups, leaves, branches, vegetation, and other litter. The result of case 1 is not as good as it could be. Even though the majority of testing data can be detected correctly, see figure 4(a). Still, the condition of lighting, shadows, and the shape of leaf pile could impact the accuracy in some scenarios and miss some objects, see figure 2-5(b) [20].

A second case study is a model training on a bigger dataset with 11 categories of litter. It shows that the accuracy could be improved by gathering more data and labelling more objects with diverse situations such as rainy, sunny, far away sight, blocked by other objects, different size of piles and different shadows that are natures of the environment on the street. Ping et al. [20] reported that different brightness could likely improve accuracy. After training with more data, overall, the results show that the performance was enhanced. Moreover, this research found that small objects like litter or leaves are difficult to identify because of the poor resolution of items.

(a) (b)

Figure 2-5 The detection results from [20] research.

Chapter 3 Research Methodology

In this section, useful techniques, dataset, modelling, evaluation metric, and experiment setup for this project will be presented in order to build and develop a model using machine learning and deep learning that enables automatically detect the presence of trash in aerial pictures.

Overall, the methodology to implement object detection using Tensorflow object detection API can be seen in figure 3-1. After obtained data from SCRAPbook. Firstly, Data-preprocessing is essential before building the model. Reviewing data that we have is one of the crucial methods since we need to know the quality and feature of it. After that, images will be annotated by using image annotation tool; this project will use LabelImg annotation tool. After the preparation step, the dataset will pass through the training step, the pre-trained CNN architectures, faster R-CNN, will be used. Then, Evaluate the model by using the mean average precision (mAP) metric. When obtaining the results, the models will be evaluated and trained to improve performance; the hyper-parameters such as the number of training step will be adjusted. Finally, making a prediction with a new image using our model to test the performance.

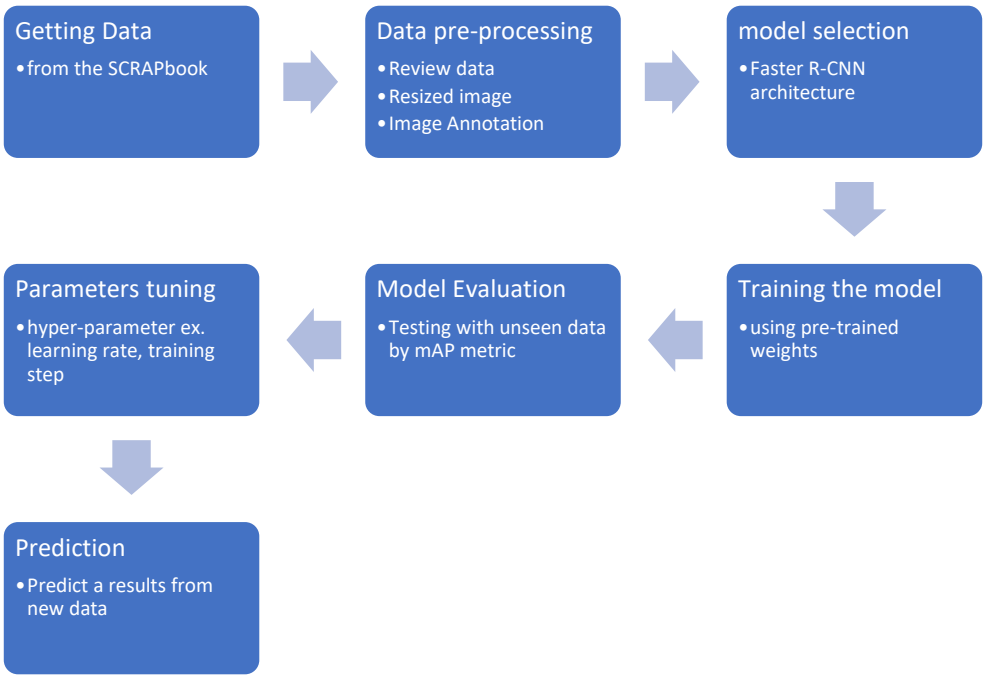


Figure 3-1 The process of implementing object detection

There are two significant experiments for the project. Firstly, an investigation about how the accuracy changes if the number of classes and labelled data are increase. Secondly, researching for the proper model that can detect litter automatically.

This project will experiment using the Python programming language in Google Colaboratory (Google Colab). The advantage of using Google Colab is that there is a GPU that users can use for free. Many libraries have been developed using Python that useful for object detection with machine learning and deep learning, such as TensorFlow, Pandas. The primary tool for this Project is Tensorflow object detection API that helps training method is easy.

3.1. Image Annotation

For supervised object detection, labelled training data or bounding boxes around objects is the vital information that allows a model can be trained in deep learning networks [22]. Labelled training data will be obtained from training data or image data by annotates interesting objects and labels them. In order to create labelled data image annotation tool is required [33]. There are many tools of image annotation, and each tool would generate different formats [33]. The most popular forms are Pascal VOL or Microsoft COCO.

In this Project, LabelImg [34] annotation tool will be used. It is open-source and effortless to use. The export format from LabelImg is PASCAL VOC, which uses XML files to store information of objects [34]. Advantages of this tool are it can draw boxes around objects and automatically save the XML files. The main reason that LabelImg is chosen because it is convenient to generate “tf record” files that will use to implement object detection in Tensorflow Object detection API [35]. Figure 3-2 illustrates samples of annotated images.

Figure 3-2 Annotated image using LabelImg

3.2. Tensorflow Object Detection API

Tensorflow object detection API [35] is an open-source tool that has been developed on Tensorflow framework. It is easy to build, train, and deploy models for object detection. It also

allows the user to create precise machine learning models that can locate and classify multiple objects in a single image that is one of the main challenges in computer vision [36].

Tensorflow API provides many pre-trained models such as SSD MobileNet, faster R-CNN resnet101, Mask R-CNN inception, etc., that allows users to avoid training a model from scratch. The numbers of architectures of Tensorflow 1 were presented in [37]

In this experiment, Tensorflow API will be used to train the model in Google Colab, which allows the user to use a free GPU. In Colab, once the model is run, it can run only a maximum of 12 hours in time, and the run-time will be restarted. However, Tensorflow API is able to save a checkpoint when the model is run [36]. This can help to keep training weights even the internet disconnects automatically.

3.3. Transfer Learning

The transfer learning method is generally used in many deep learning problems, and it has been performed successfully in many applications, especially in CNN architectures [38]. Geron [22] suggest that transfer learning can help speed up training as well as require a much smaller dataset. This method will take a pre-trained model with pre-trained weights that help to reduce the training time. However, training a CNN from pre-trained model requires the modification of some hidden layers and the final classification layer related to our data.

If the number of data or images is relatively small, it is not an excellent way to develop a huge CNN from scratch because parameters and weights will be trained from unknown value. Besides, A large amount of labelled training data are needed to ensure that a proper model convergence.

Since Scrapbook training image that contains litter is quite small, transfer learning is beneficial for this project. A pre-trained weight of Faster R-CNN models that are trained on COCO dataset will be used instead of training from scratch.

The COCO dataset is a large image data for object detection and image segmentation developed by Microsoft [20]. The data collection consists of different types of small objects with the context complexity, so it is ideal for the identification of small objects. This form of transfer learning allows the network to converge more easily and lowers the chances of overfitting because our data set is limited.

3.4. Faster R-CNN architecture

This section will introduce the critical key of Faster R-CNN that is the main model for implementing object detection for this project. Faster R-CNN would be taken to use for litter coastal image analysis in Scotland because it can provide high performance with high-speed training for object detection for many studies [32].

In these days, one of the most widely used R-CNN architecture for object detection is the Faster R-CNN that was developed from Fast R-CNN. The primary approach that allows Faster R-CNN works better and faster is the introduction of RPN, which is a fully convolutional neural network. It will enable the model can train end-to-end to classify boundaries of objects into categories or background. [8] said that RPN is a vital key for Faster R-CNN that makes it is one of the best frameworks for scholars.

Faster R-CNN technique consists of two components. The first component is an RPN that uses to create object proposals. The second network is the fast R-CNN detector that uses to classify the class of the object.

Faster R-CNN architecture is shown in figure 3-3. The original paper for more deeply details can be found in [8]. Instead of Selective Search, Faster R-CNN uses RPN layers to create region proposals from feature maps. RPN uses the sliding window on the feature map and extracts candidates for an object. Each object in an image has bounding boxes, called anchor boxes, in order to draw positions of objects. Then, the region proposals are located into the feature map, and objects are detected by classifying region proposals. Since Faster R-CNN produces image features and region proposals using a single CNN, in addition to faster detection, there is a benefit that end-to-end training can be done.

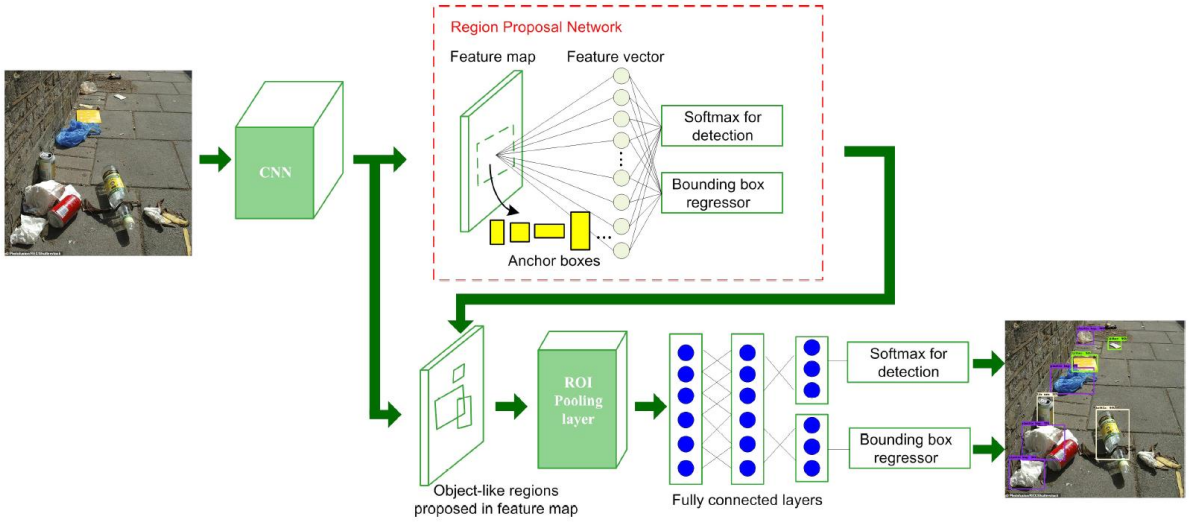


Figure 3-3 The Fast R-CNN architecture [8]

In order to perform classification, detection, or segmentation for the proposed objects in this architecture, Faster R-CNN composed of some essential components.

- **Region Proposal Network (RPN)**

RPN can deal with the problem of the unknown number of objects that the model needs to detect. it aims to produce a collection of proposals; each has a score of its probability to be an object as well as its class or label of the object [8]. To accomplish this task, RPN will take any size of input data. Then, these proposals will be passed to 2 fully connected layers, which one is bounding box regression and another one is bounding box classification, i.e. the object part or background part.

This architecture uses a nxn sliding window across the feature map. Every sliding window will be mapped to a lower dimension window. The sliding window also provides the location of an object in the image.

- **Anchor Boxes**

Anchor Boxes are a crucial component for Faster R-CNN. The sliding window from RPN approach, also known as anchor boxes. They are used to identify the class or label of each object of the bounding box and are typically selected based on object sizes in the training data. They are typically centred of the sliding window [8].

The main purpose of using Anchor boxes is that it can predict multiple objects at once, detect different sizes of objects, and overlapping objects. Girshick et al. [8] present that this can help speed up close to real-time prediction and improve the performance of object detection.

3.5. Litter Detection using Faster R-CNN model

In section 2.4.3, it explained that Faster R-CNN is the best option if very high accuracy is necessary but could be slower in training time compared with other architectures. Thus, Faster R-CNN was selected for this project. However, there are different kinds of convolutional feature extractors that used for Faster R-CNN, such as Inception v2, VGG and Resnet101. The convolutional feature extractor is a crucial method to obtain high-level features, but it directly affects speed and performance of the model because different feature extractors would contain the different number of parameters and types of layers [39]. Different kinds of feature extractor would appropriate for different data.

In 2017, J. Huang et al. [39] researched the comparison of the performance of each feature extractor. They trained Faster R-CNN with 6 feature extractors that are VGG, Resnet101, Inception v2, inception v3, Inception Resnet, and MobileNet, which each will be explained below.

- VGG [40]: the features are extracted from the “conv5” layer whose stride size is 16 pixels. It is cropped and resized feature map to 14x14 and maxpool to 7x7.
- Resnet 101 [41]: the features are extracted from the last layer of the “conv4” block with the stride size is 8 pixels when operating in atrous mode, and 16 pixels for other modes. They crop and resize feature maps to 14x14 and maxpool to 7x7.
- Inception v2 [42]: They extracted features from “Mixed_4e” layer with the stride size is 16 pixels. Feature maps are cropped and resized to 14x14.
- Inception v3 [43]: The features are extracted from the “Mixed_6e” layer with the stride size is 16 pixels and feature maps are cropped and resized to 17x17.
- Inception Resnet [44]: The features are extracted from the “Mixed_6a” layer, including residual layer. The stride size is 8 pixels when operating in atrous mode, and 16 pixels for other modes. Feature maps are cropped and resized to 17x17.
- MobileNet [45]: The features are extracted from the “Conv2d_11” layer with the stride size is 16 pixels, and feature maps are cropped and resized to 17x17.

The results from [39] research that is useful for our project is about the comparison of accuracy for the small size of the object because most of the items in our dataset are small sizes. Figure

3-4 demonstrates the comparison of performance by object size for CNNs models with different feature extractors. It is evident that Faster R-CNN obtained higher mAP (small) and mAP (medium), see section 3.6, for several feature extractors. The most top mAP for small and medium size of objects seem to be Faster R-CNN Inception Resnet v2 and followed by Faster R-CNN Resnet101.

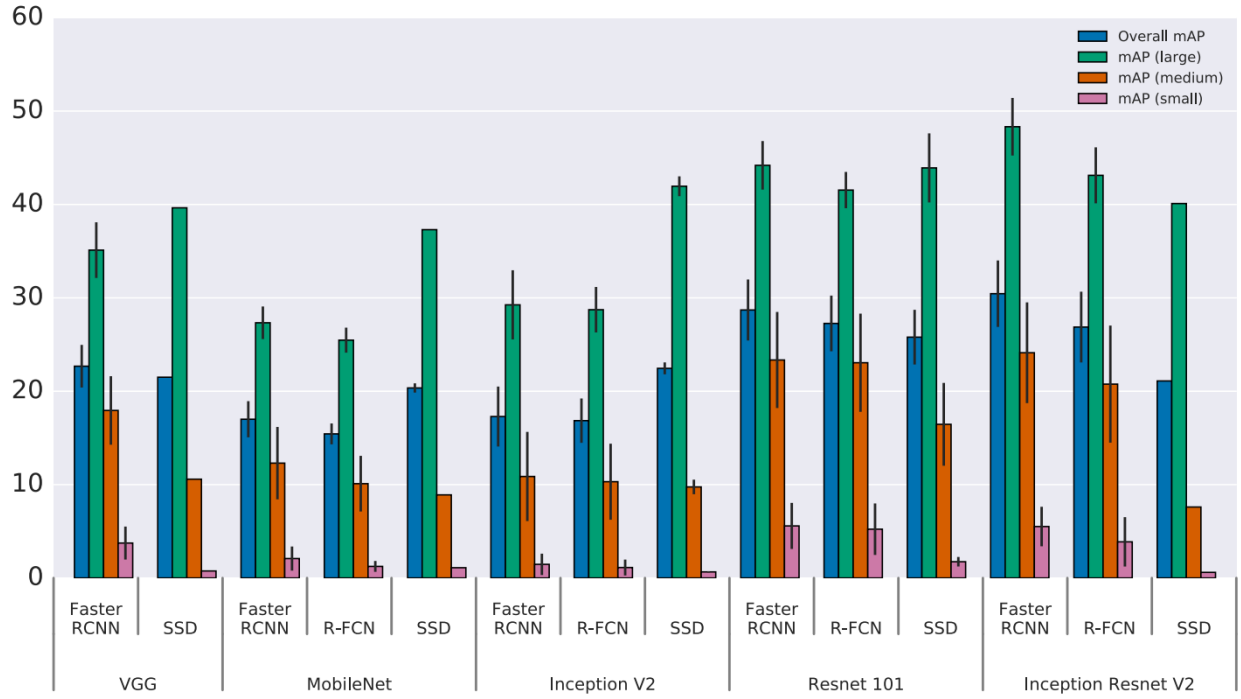


Figure 3-4 the comparison of accuracy by object size for each CNN architectures using different feature extractors [39].

However, the period of training model should be considered as well. Figure 3-5 shows the comparison between speed and accuracy in different CNN models and feature extractors [39]. It can be regarded that training time and accuracy would have direct variation in every model. Even though Faster R-CNN Inception Resnet v2 could provide the highest efficiency, but it would also spend the most training time. In contrast, R-CNN Inception v2 would spend the least training time and still generate an acceptable accuracy.

As a result, in this project, R-CNN Inception v2 is chosen to investigate how the performance changes by varying the number of labelled images and classes. Then, it will be compared with Faster R-CNN Inception Resnet v2 and Resnet 101 to research for the excellent results.

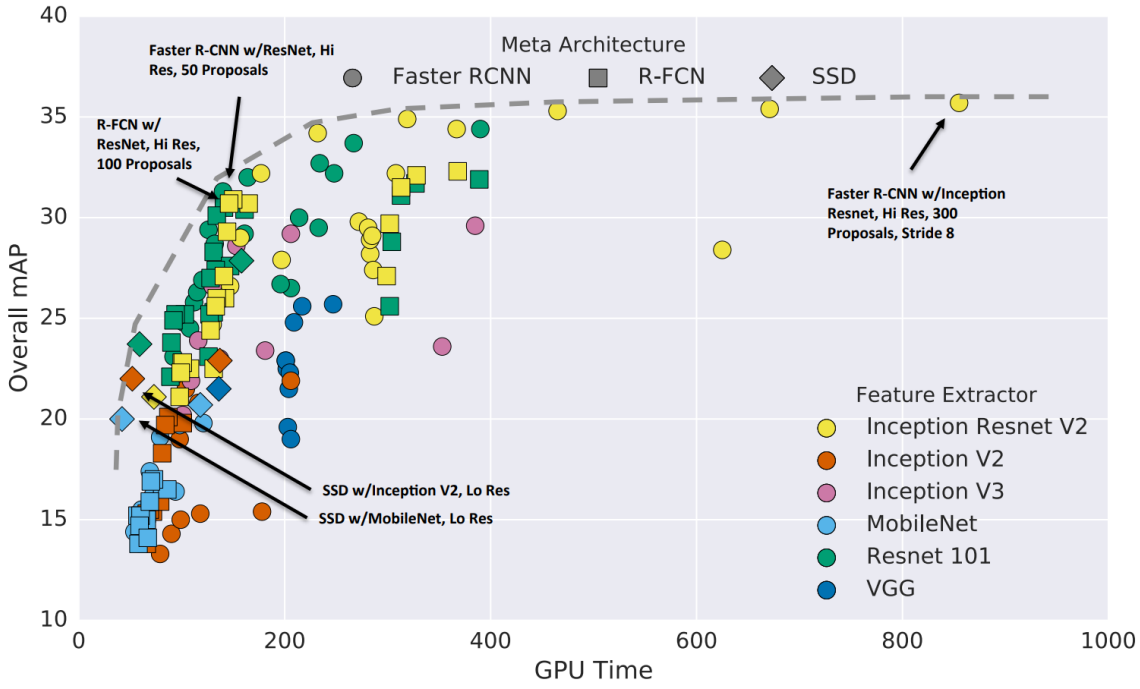


Figure 3-5 Accuracy vs training time of different CNN architectures and feature extractors [39].

3.6. Evaluation Metric

An evaluation metric is a useful tool in order to evaluate, analyse, and compare the results.

3.6.1. Confusion Matrix

A confusion matrix is a chart that is widely used in machine learning and deep learning problem. It is used to show a summary result and define the performance of a classification model or prediction model on a testing data that actual value are labelled. The number of predictions that are correct and incorrect is summarised with count values and broken down by class. A confusion matrix can be used to compute recall, precision and accuracy. Table 3-1 represents the Confusion Matrix chart where TP is a true positive value, TN is a true negative value, FN is a false negative value, and FP is a false positive value

Table 3-1 Confusion Matrix chart

	Predicted: no litter	Predicted: litter
Actual: no litter	TN	FP
Actual: litter	FN	TP

“Accuracy” is a metric that measures how a model predicts accurately show in eq.1. However, this assumes equal weight for both kinds of error. Sometimes, 99% of accuracy can be poor, depending on the problem. Thus, Precision, Recall and F-measure score could help.

$$Accuracy = \frac{TN+TP}{TP+TN+FP+FN} \quad \text{eq.1}$$

“Precision” is a value when the total number of samples are correctly predicted, and the total number of examples are predicted as a positive [22]. The higher precision is a higher chance for a sample classified as positive to actually be positive. The equation will be given in eq. 2, where TP is a true positive number, and FP is a false positive number.

$$Precision = \frac{TP}{TP+FP} \quad \text{eq.2}$$

“Recall” is the number of actual positive class that was predicted correctly of samples in the dataset [22]. High recall would provide low precision. High recall implies that the model gives the most relevant results. The equation will be given in eq. 3, where FN is a false negative number.

$$Recall = \frac{TP}{TP+FN} \quad \text{eq.3}$$

However, it is difficult to compare two models that have a large different value of Recall and Precision, so “F-measure” have been discovered to compare them. It helps to calculate Recall and Precision at the same time. Eq.4 shows the equation of F-measure.

$$F - measure = \frac{2xRecallxPrecision}{Recall+Precision} \quad \text{eq.3}$$

3.6.2. Mean Average Precision (mAP)

Mean Average Precision (mAP) is a standard metric used in object detection problems [22]. This metric is motivated by the method of precision and recall by computes the area under the curve of Precision/Recall curve [22].

The model performance can be evaluated by calculating the maximum precision [22]. For example, it can be obtained at least 0% recall, then 10%, 20%, until 100%, and then compute the mean of maximum precisions, called Average Precision (AP) metric. However, if the classes are more than two classes, the mean of AP value from each class can be calculated, known as mAP.

Nevertheless, there is an extra degree of difficulty in an object detection task. Sometimes the model predicted class correctly, but at the wrong location, and this should be a positive prediction. IoU threshold can be solved this problem. Intersection over Union or IoU is the overlap region of bounding boxes between the predicted and target box, divided by the area of their union, see figure 3-6 [22]. For instance, the prediction would be correct only when the IoU greater than some value, and the class is predicted correctly. It generally presents this metric as AP@0.5 that means Average Precision (AP) with intersection over union (IoU) threshold is 0.5. In Tensorflow object detection API, mAP with IoU thresholds ranging between 0.5 to 0.95 with 0.05 increments [20].

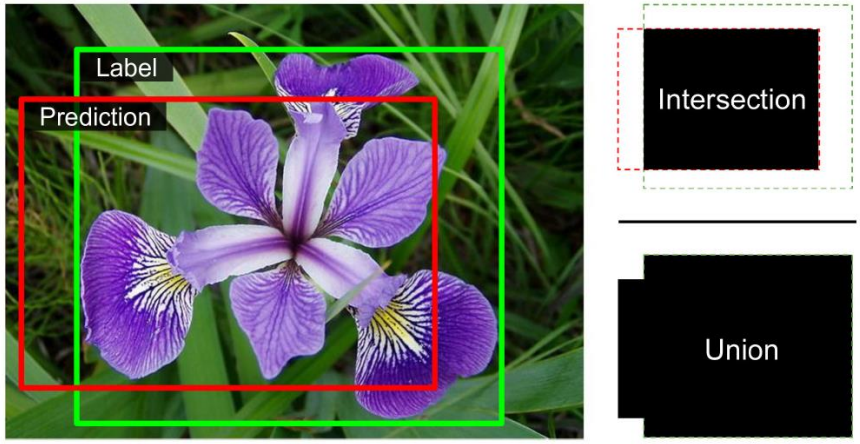


Figure 3-6 IoU metric for object detection [22].

3.6.2.1. COCO challenge metric

In this project, the metric from COCO competition [9] was adopted from mAP metric will be used to evaluate the results. Standard mAP metric used an IoU threshold of 0.5, but the COCO metric defines mAP metric with other thresholds. The definition of each parameter will be explained as following [46].

- DetectionBoxes_Precision/mAP means the average precision over all images, classes 10 IoU thresholds ranging from 0.5 to 0.95 with 0.05 increments.
- DetectionBoxes_Precision/mAP@.50IOU describes the average precision using only IoU threshold = 0.5 but still compute over all images and classes.
- DetectionBoxes_Precision/mAP@.75IOU describes the average precision using only IoU threshold = 0.75 but still compute over all images and classes.

The COCO metric also computes across different object sizes. This can provide the performance in different object scales.

- DetectionBoxes_Precision/mAP (small) is the average precision for all small objects or small bounding boxes, which the area is below 32x22 pixels.
- DetectionBoxes_Precision/mAP (medium) means the average precision for all medium-sized of objects or bounding boxes, which the area is between 32x32 pixels and 96x96 pixels.
- DetectionBoxes_Precision/mAP (large) describes the average precision for every large object, which the area is between 96x96 pixels and 10,000x10,000 pixels.

Similar to the mAP, AR metrics will be varied different numbers of detections in an image and the size of a detected object.

- DetectionBoxes_Recall/AR@1 means the average recall across all images with at most 1 detection.
- DetectionBoxes_Recall/AR@10 is the average recall across all images with at most 10 detections.
- DetectionBoxes_Recall/AR@100 means the average recall across all images with at most 100 detections.
- DetectionBoxes_Recall/AR@100 (small) is the average recall for small objects with at most 100 detections.
- DetectionBoxes_Recall/AR@100 (medium) means the average recall for medium objects with at most 100 detections.
- DetectionBoxes_Recall/AR@100 (large) means the average recall for large objects with at most 100 detections.

3.7. Experimental

The experiments will use the dataset from SCRAPbook project, which is aerial images of litter on Scottish's coastal. Firstly, in experiment 1, this project will investigate how the performance changes if the number of labelled images and classes are increased. In order to investigate this, it will be divided in 6 cases, which are 1 class of object with 20 training images, 100 training images, and 500 training images and 6 classes of objects with 20 training images, 100 training images, and 500 training images. In 1 class experiment, it will be litter, while 6 classes will be

litter, person, building, vehicle, boat, and tire. Each case will be trained using Faster R-CNN inception v2 and 20000 training steps, which would be enough to make the loss converge close to zero. Then, the results will be evaluated using coco metric. More detail for the experimental setup for each case can be found in Appendix E.

In experiment 1, one class was used to experiment because the aim of this project is to detect litter and objects in each image is relatively small, including they are low resolution and quality. Even using our eyes are still difficult to classify and annotate each type of litter, such as plastic, paper, etc. Thus, all sorts of litter were identified as litter class. However, there are many incorrect detections for other small objects, such as boats or vehicles, so 6 classes would be experimented to compare the performance of the results. For the 6 classes experiment, other kinds of objects that usually appear and missed detection in our dataset were annotated into their own classes, including litter, building, person, vehicle, boat, and tire. Actually, the experimenter would like to classify types of litter, such as plastic bac, plastic bottle, glass bottle, etc., but as aforementioned, we could not identify these class because the objects are too small.

Turning to the experiment 2, the project will examine the appropriate model for our dataset by training the model with other pre-trained models that are Faster R-CNN Inception Resnet v2 and Faster R-CNN Resnet 101 COCO, which perform outstandingly for small object detection. Then compare the performance with Faster RCNN inception v2. This experiment will use the 6 classes with 500 images to train models because it could outperform others. Each model will be trained until the loss nearby 0.1. This experiment will use the COCO metric to evaluate each model. In addition, these three models will be used to predict the unseen data with 100 images that contain litter and no litter in the images and focus only litter class. It means that if there is litter appears in an image (even only one piece), so it will be labelled as litter image (1), but if there is no litter, it will be labelled as no-litter (0). Then, comparing by using the confusion metric to compute Recall, Precision, Accuracy, and F-measure.

3.7.1. Dataset

Our image dataset obtained from SCRAPbook project [6] that captured from the small aircraft. They are images of coastal in Scotland with litter. The aerial picture resolution is high enough to identify an object greater than 10 centimetres, but the image resolution and quality is quite poor. This dataset consists of 11,693 images in total, but not every picture would contain litter. It includes various types and conditions including sunny/cloudy, far away, the coastline with the city, mountain view. There are many types of objects that appear, such as a person, house,

building, car, van, truck, tire, sheep, plastic, other litter, etc. The samples of images in dataset are provided in figure 3-7. However, the challenge in this data is some objects in an image is extremely small with a low quality. Sometimes even my eyes cannot identify the class of the item that appears in the picture.

Figure 3-7 Sample images in the dataset

Nevertheless, the quality of some images is not good to train the model. Finally, 500 images that contain litter were selected and resized to 800x600 pixels. In order to prepare the training method, the dataset is divided into 80% training image and 20% testing image of each case study. Since a small amount of data may give a poor performance, data augmentation method will be used to generate more sample from existing images. In addition, the Transfer learning approach is able to tackle when the dataset is relatively small.

In experiment 1, the number of classes will be 1 class and 6 classes, and each will be tested on 20 images, 100 images and 500 images. The amount of labelled training data and labelled testing data of 1 class and 6 classes are provided in table 3-2 and 3-3, respectively.

The experiment 2 will use the 6 classes with 500 images, which contains around 1,800 labelled objects, to train the model on Faster R-CNN Inception v2, Faster R-CNN Inception Resnet v2 and Faster R-CNN Resnet 101 COCO.

3.7.2. Experimental Setup for Google Colab

This project will be trained on Google Colaboratory or Colab and investigated on TensorBoard. The Colab machine using Nvidia Tesla K80 GPU and users can use it for free [47]. GPU allows us to reduce the running time compared to training it on the local machine with a single CPU.

Table 3-2 Dataset for training and testing one class of object in different number of images

Number of images	class	Labelled training data	Labelled testing data	Object size
20	litter	100	48	Small to medium
100	litter	287	73	Small to medium
500	litter	1424	229	Small to medium

Table 3-3 Dataset for training and testing six classes of object in different number of images

Number of images	class	Labelled training data	Labelled testing data	Object size
20	boat	22	9	Small to medium
	building	41	7	Small to large
	litter	89	5	Small to medium
	person	6	4	small
	tire	8	1	small
	vehicle	49	7	Small to medium
100	boat	27	9	Small to medium
	building	51	16	Small to large
	litter	344	72	Small to medium
	person	13	5	small
	tire	9	3	small
	vehicle	54	22	Small to medium
500	boat	62	11	Small to medium
	building	157	18	Small to large
	litter	1400	223	Small to medium
	person	33	7	small
	tire	22	6	small
	vehicle	144	23	Small to medium

Note: small size < 32x32 pixels,

32x32 < medium size < 96x96,

96x96 < large size < 10,000x10,000

Chapter 4 Result and Discussion

4.1. Experiment 1

4.1.1. Result of Experiment 1

The results obtained from training models using the different number of class and images for 20,000 iterations. The numbers of classes were 1 class and 6 classes, and each was tested on 20 images, 100 images and 500 images. One class is only litter, and six classes are included litter, building, person, vehicle, boat, and tire. The result will be analysed, evaluated, and compared in section 4.1.2.

Figure 4-1, 4-2 and 4-3 show the correct detection, missed some objects, and incorrect detection, respectively, of one class with 20 training samples. This case study used a small number of training images to train the model. The results show that even some litter are detected correctly, but there are many incorrect classifications appear in prediction images. Figure 4-2 illustrates there are missed some litter, and figure 4-3 classifies cars as litter.

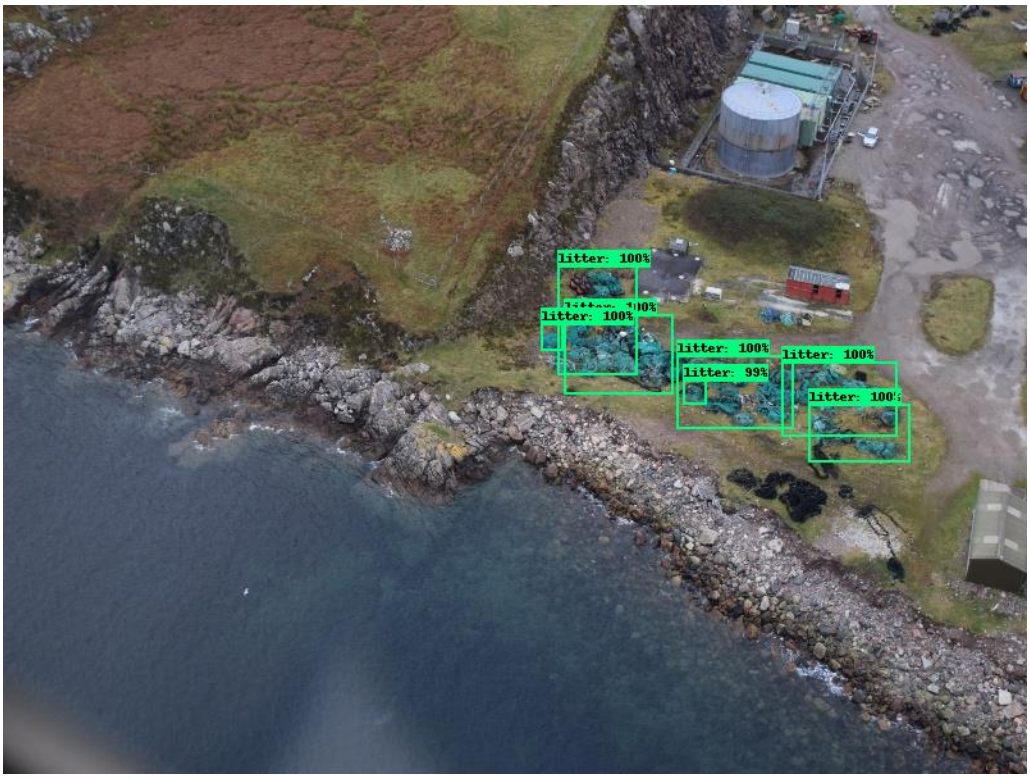


Figure 4-1 Correct detections from one class with 20 training images

Figure 4-2 Missing detect some objects of one class with 20 training images

Figure 4-3 Incorrect detections from one class with 20 training images

Figure 4-4, 4-5 and 4-6 demonstrate the correct detection, missed some objects, and incorrect detection, respectively, of one class with increase training images to 100. The results show that most of the litter are classified correctly, but some litter are missed detected in figure 4-5. Moreover, there is classification incorrectly in figure 4-6 that pebbles were recognised as litter.

Figure 4-7, 4-8 and 4-9 show that most of the litter were detect correctly, and there are fewer wrong class detections for the model of one class with 500 training images. Moreover, figure 4-7, which is the same image as figure 4-3, shows that no detection box is the correct classification. However, a few litters miss detections in figure 4-8.

Figure 4-4 Correct detections from one class with 100 training images

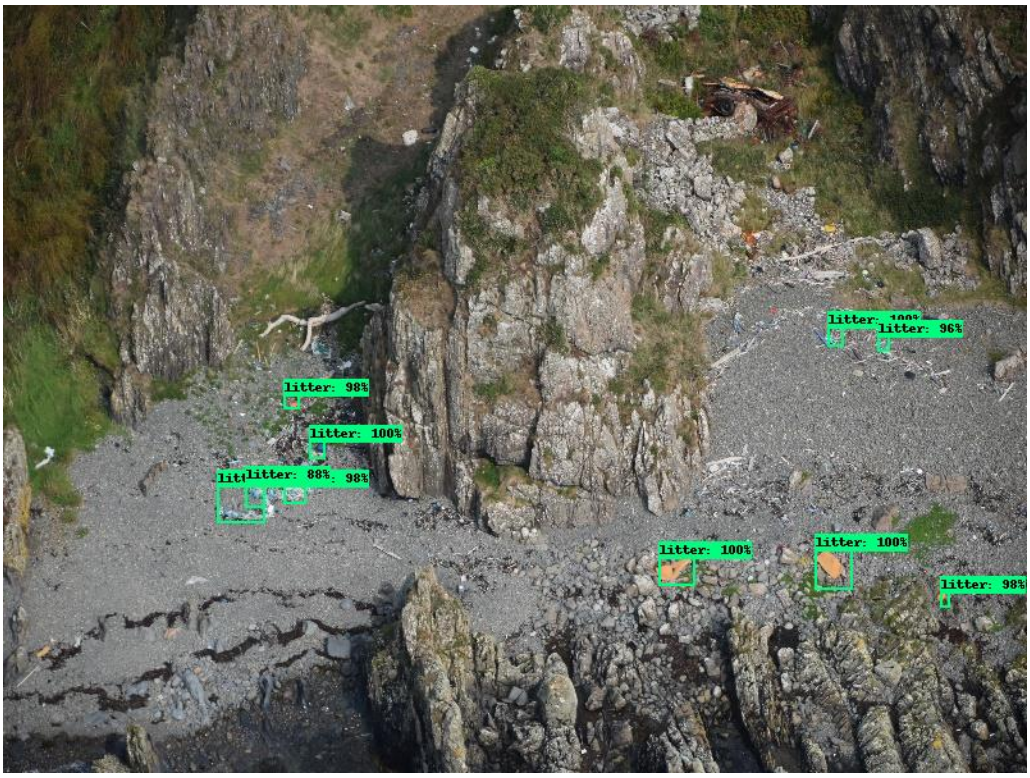


Figure 4-5 Missing detect some objects of one class with 100 training images

Figure 4-6 Incorrect detections from one class with 100 training images

Figure 4-7 Correct detection from one class with 500 images

Figure 4-8 Missing detect some litter from one class with 500 training images

Figure 4-9 Incorrect detection from one class with 500 images

For 6 classes experiment, figure 4-10, 4-11 and 4-12 show the correct detection, missed some objects, and incorrect detection, respectively, of six classes with 20 training samples. This case study used a small number of training images to train the model. Overall, this model looks to perform well because most of the objects were detected correctly in figure 4-10 and 4-11; even there are error detections in figure 4-11 and 4-12.

Figure 4-13, 4-14 and 4-15 show the correct detection, missed some objects, and incorrect detection, respectively, of six classes with 100 training samples. This model seems better for litter detection that most of the litter were detected. Nevertheless, figure 4-15 show incorrect classifications that boats were classified as litter that the previous model does not wrong.

Lastly, figure 4-16, 4-17 and 4-18 demonstrate the sample results that correct detection, missed some objects, and incorrect detection, respectively, of six classes with 500 training samples. Overall, most of the litter in each image were detect correctly. However, the pile of branches was not identified in figure 4-17, and some boats were classified as litter in figure 4-18.

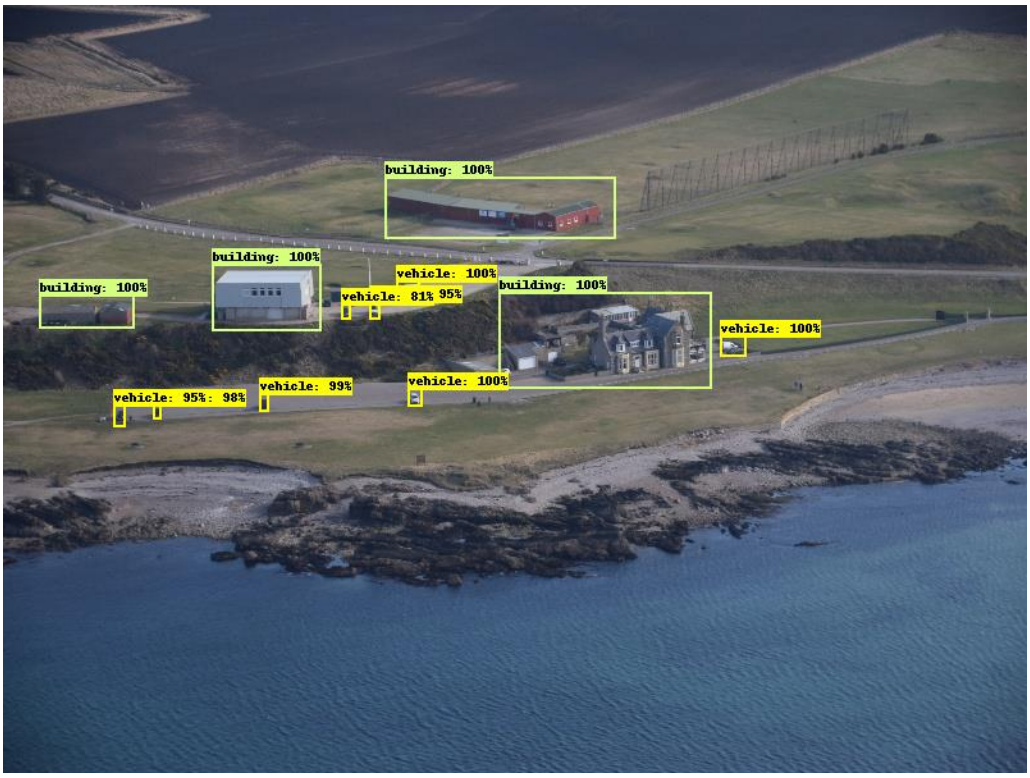


Figure 4-10 Correct detection from six classes with 20 images

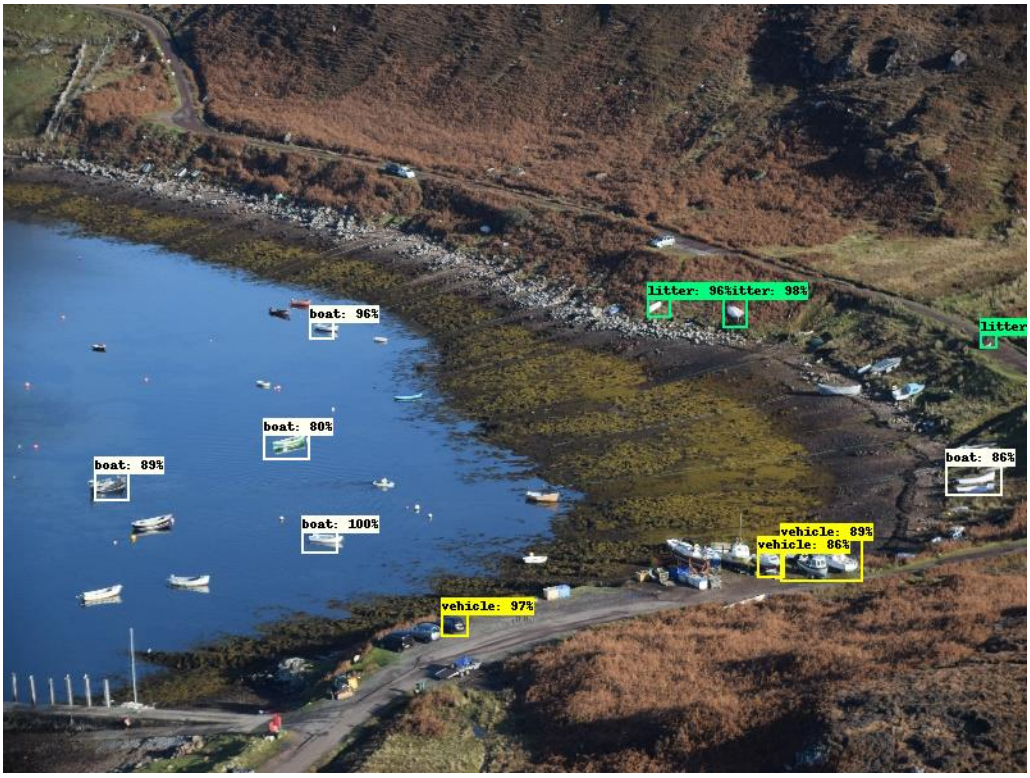


Figure 4-11 Missing detect some objects from six classes with 20 images

Figure 4-12 Incorrect detection from six classes with 20 images

Figure 4-13 Almost correct detection from six classes with 100 images

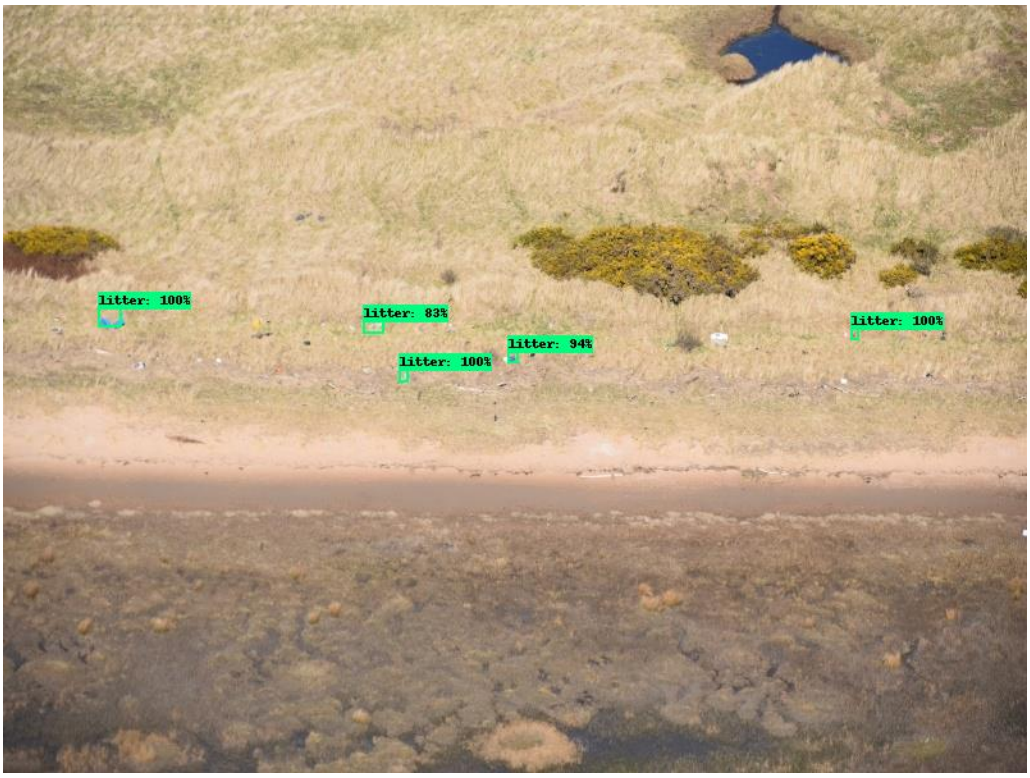


Figure 4-14 Missing detect some litter from six classes with 100 images

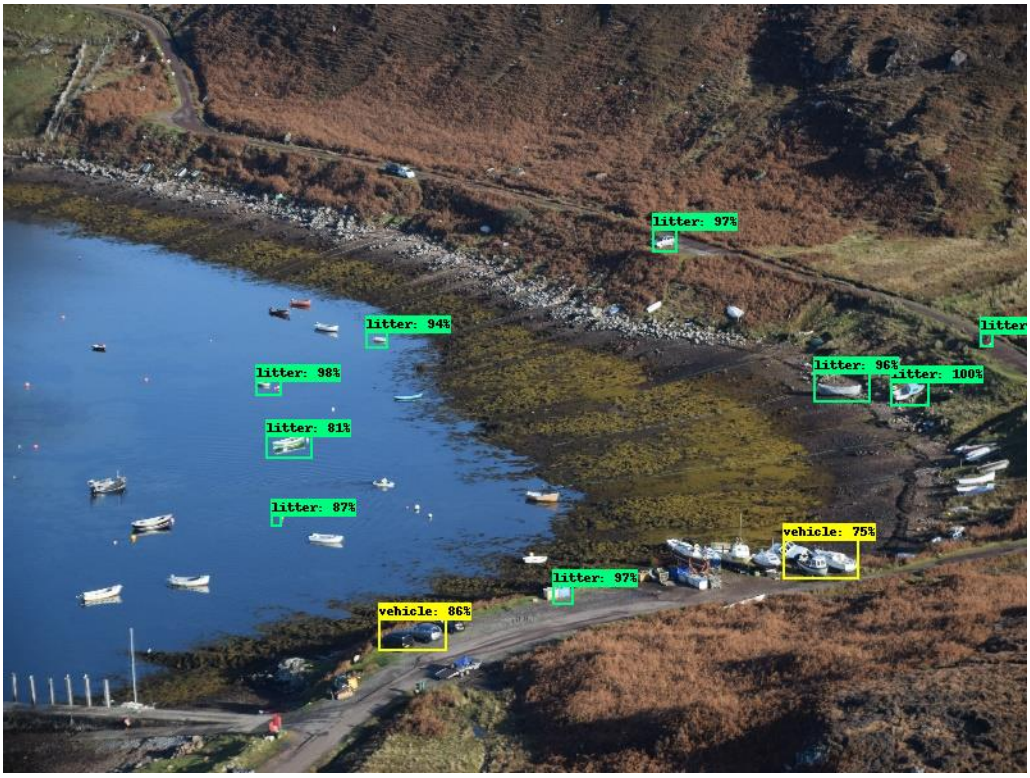


Figure 4-15 Incorrect detection from six classes with 100 images

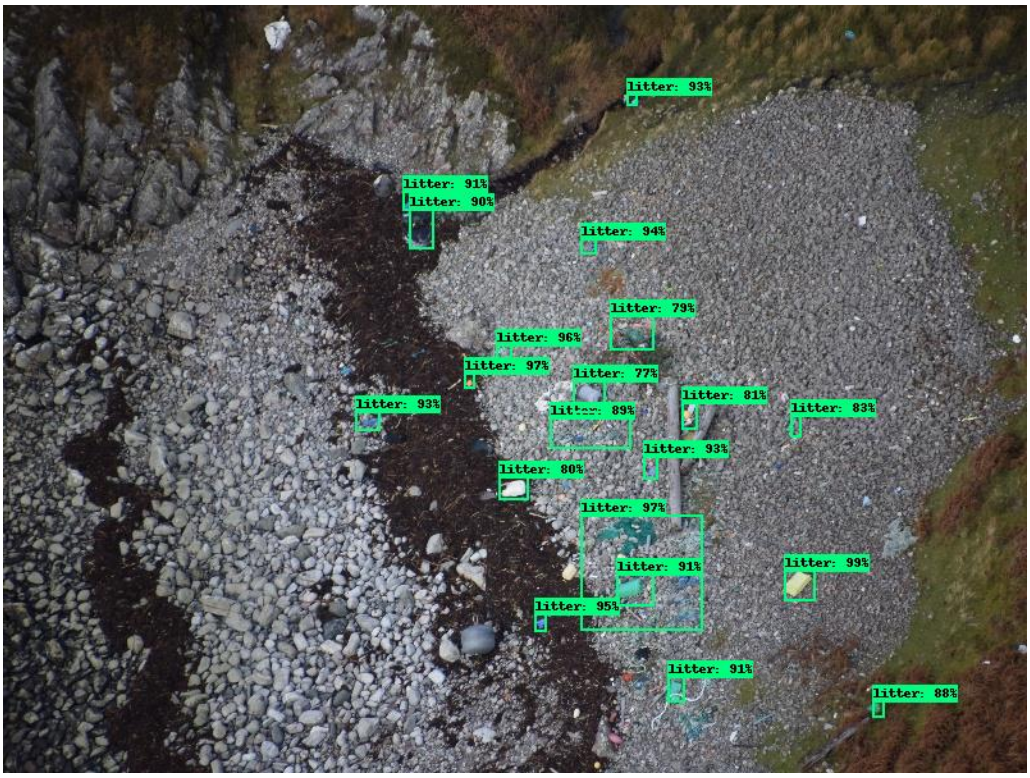


Figure 4-16 Almost correct detection from six classes with 500 images

Figure 4-17 Missing detect some litter from six classes with 500 images

Figure 4-18 Incorrect detections from six classes with 500 images

Figure 4-19 to figure 4-24 demonstrate the results of each model that evaluate by using COCO metric based on mAP. Each figure shows the results of mAP, mAP (large), mAP (small), mAP@.5IoU, and mAP@.75IoU. The X-axis describes the number of training steps, and the Y-axis represents the mAP score. The description of each metric can be seen in section 3.6. Overall, the results show that when the training steps are gone up, mAP score seems to increase. However, it can be noticed that some values were -1 for mAP (large), these mean there is no large objects (96x96 pixels < area < 10,000x10,000 pixels) in the training data, especially for models with one class.

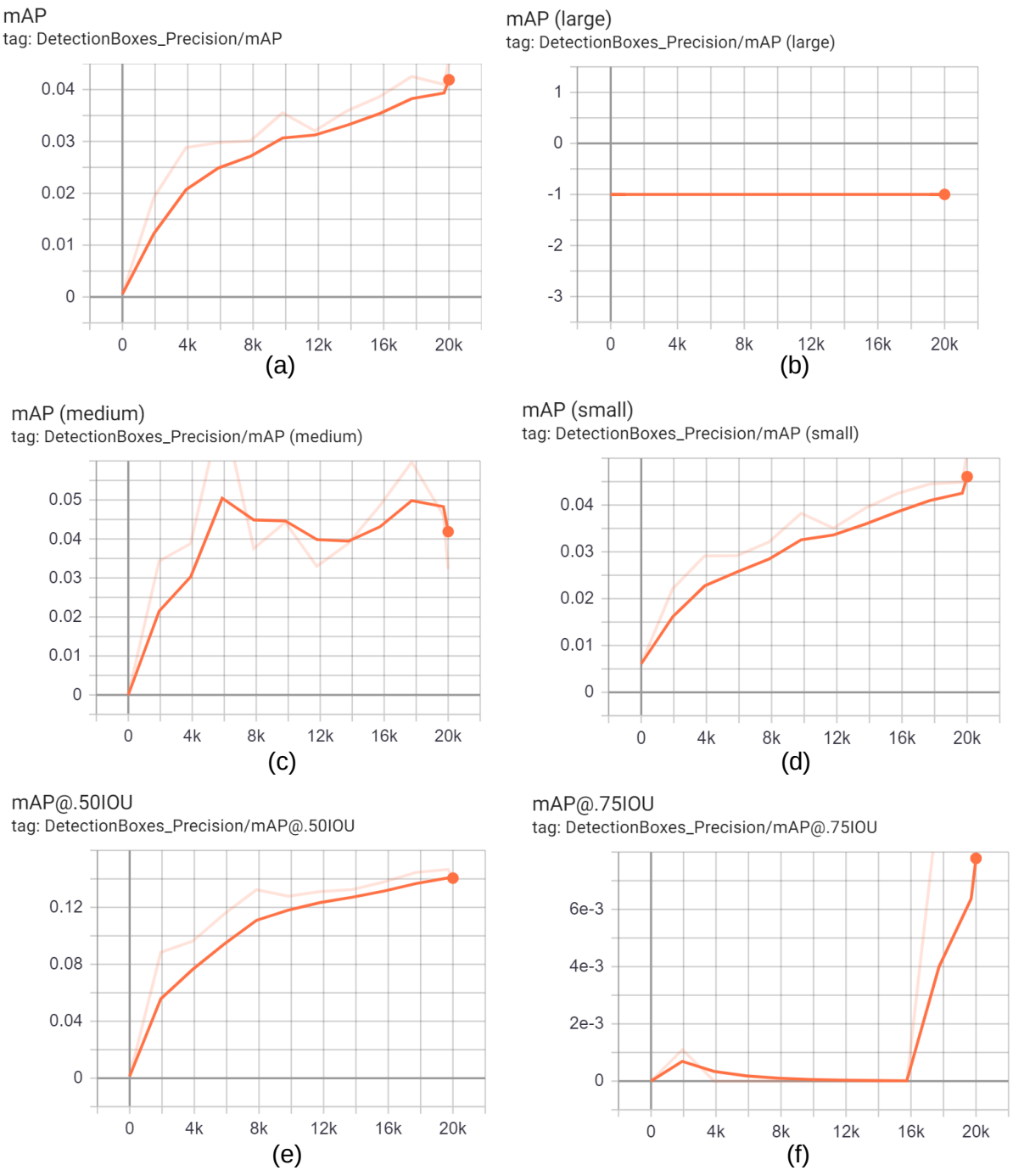


Figure 4-19 The results of 1 class with 20 images at 20,000 step using mAP metric

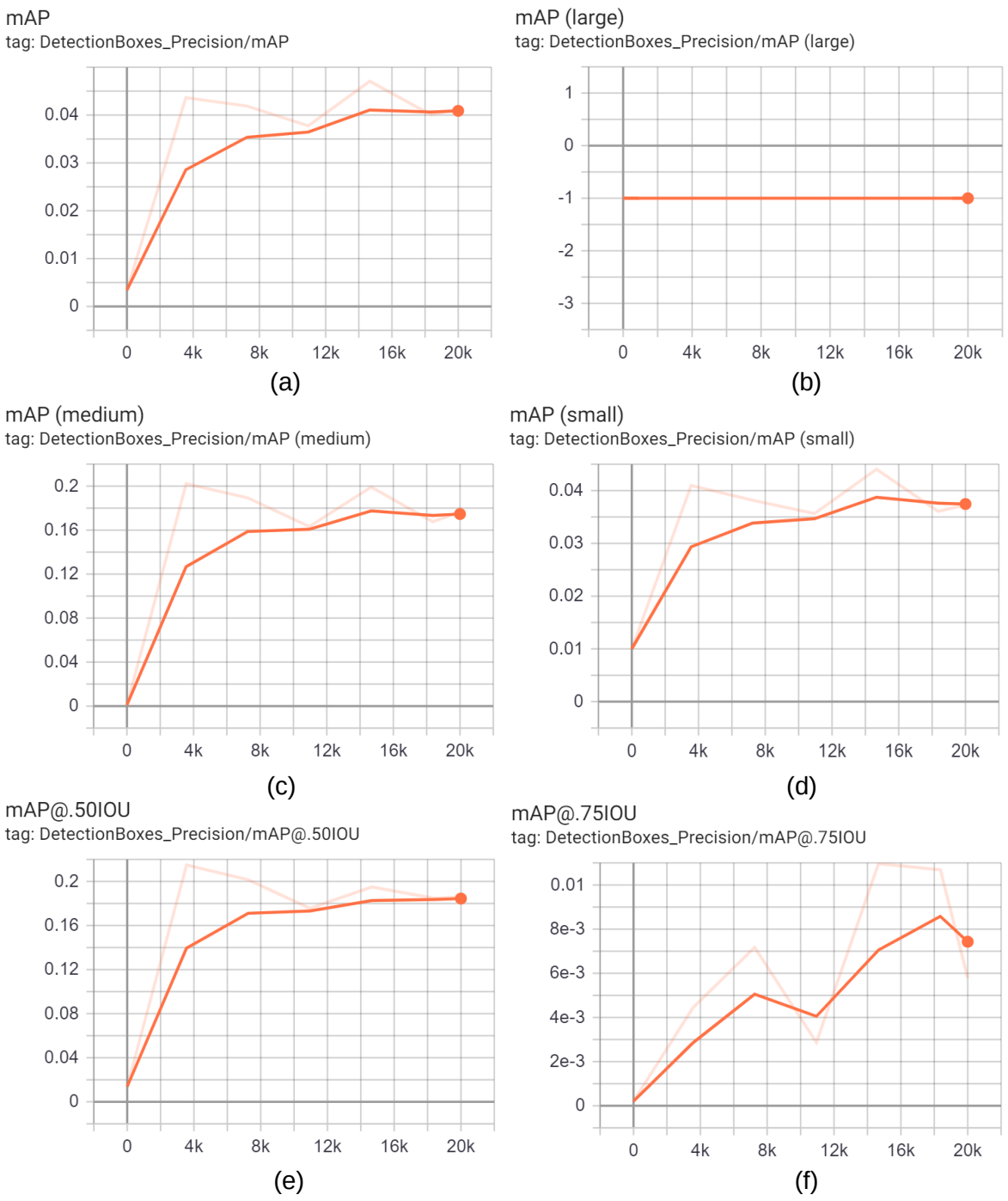


Figure 4-20 The results of 1 class with 100 images at 20,000 step using mAP metric

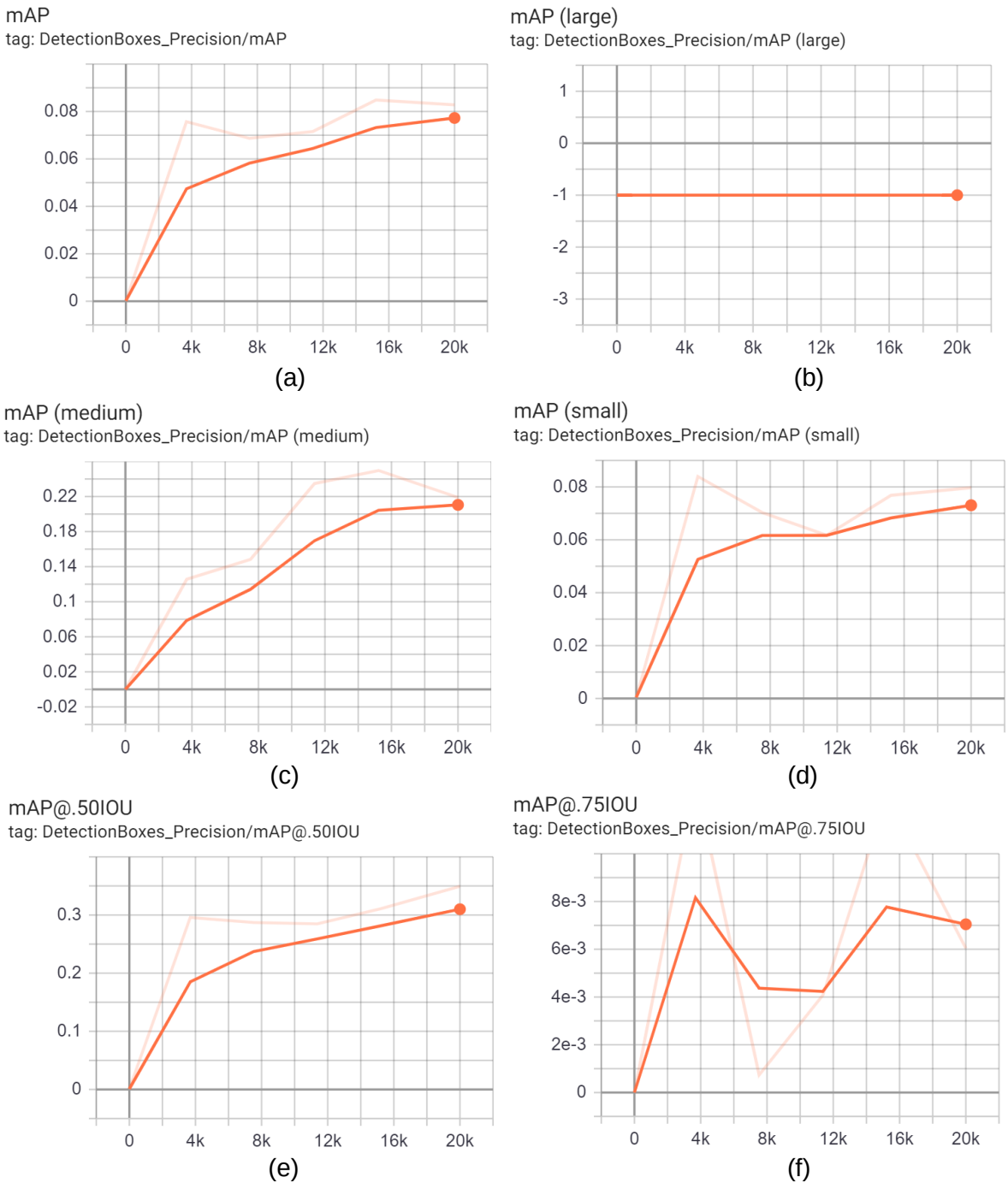


Figure 4-21 The results of 1 class with 500 images at 20,000 step using mAP metric

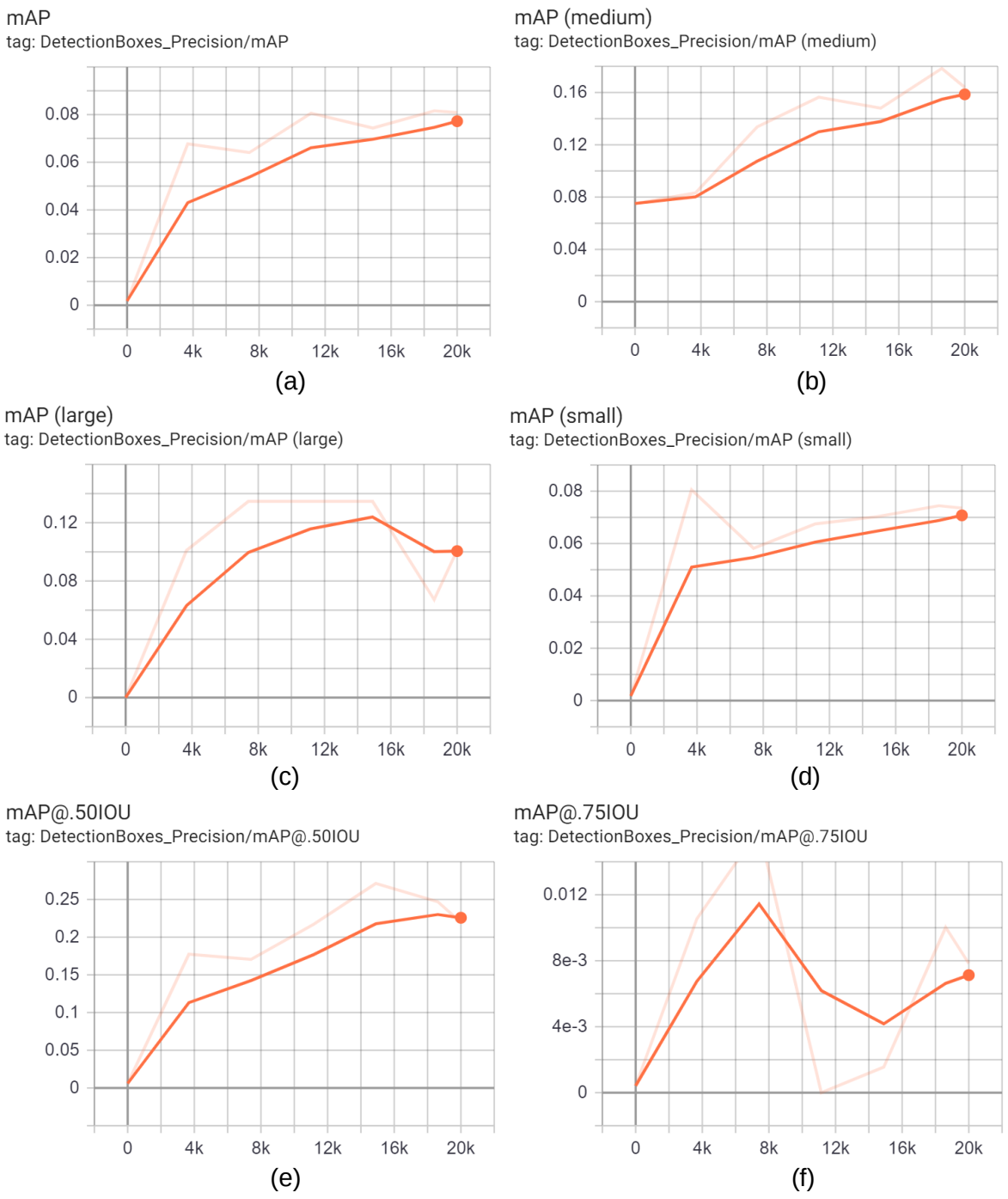


Figure 4-22 The results of 6 classes with 20 images at 20,000 step using mAP metric

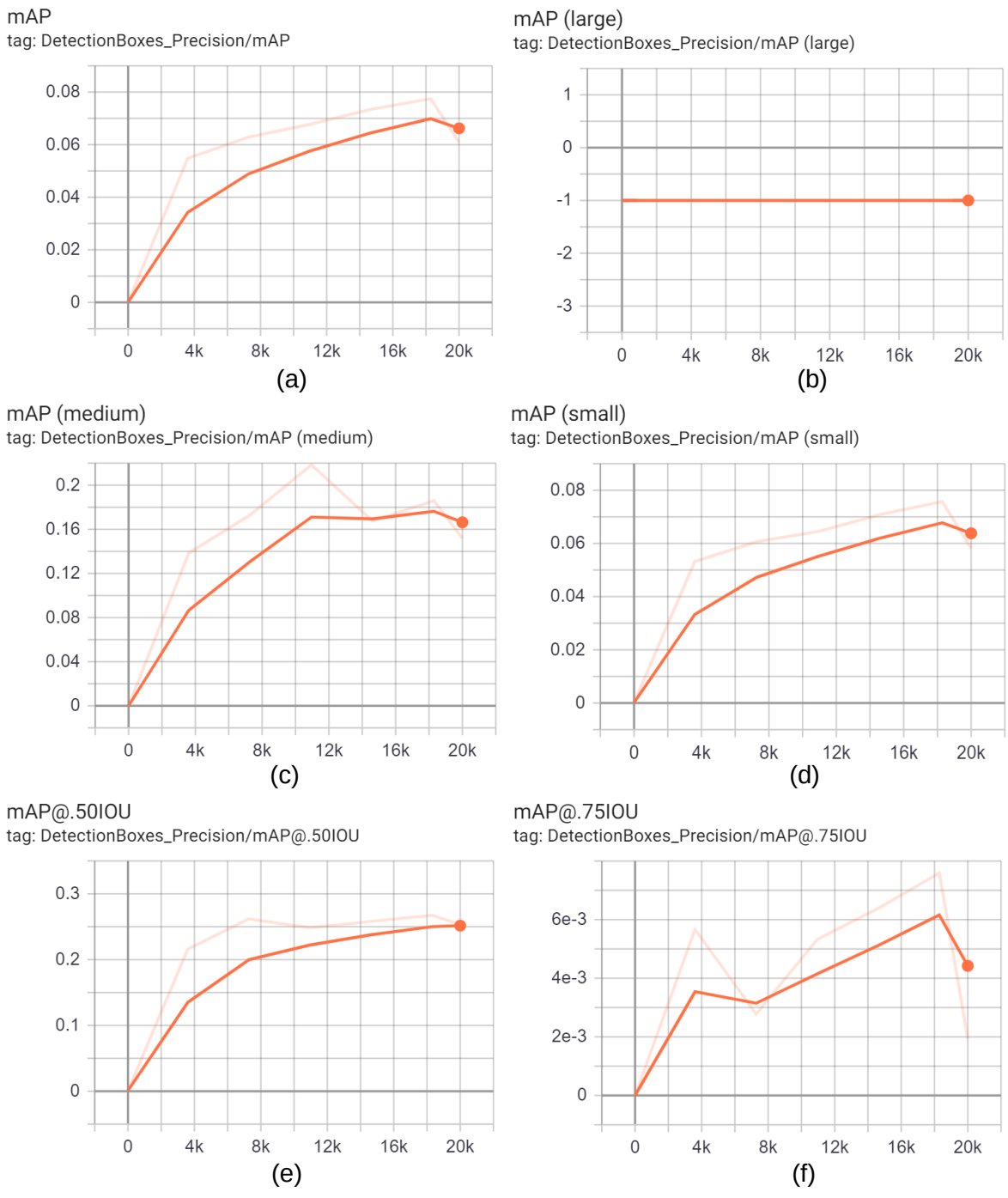


Figure 4-23 The results of 6 classes with 100 images at 20,000 step using mAP metric

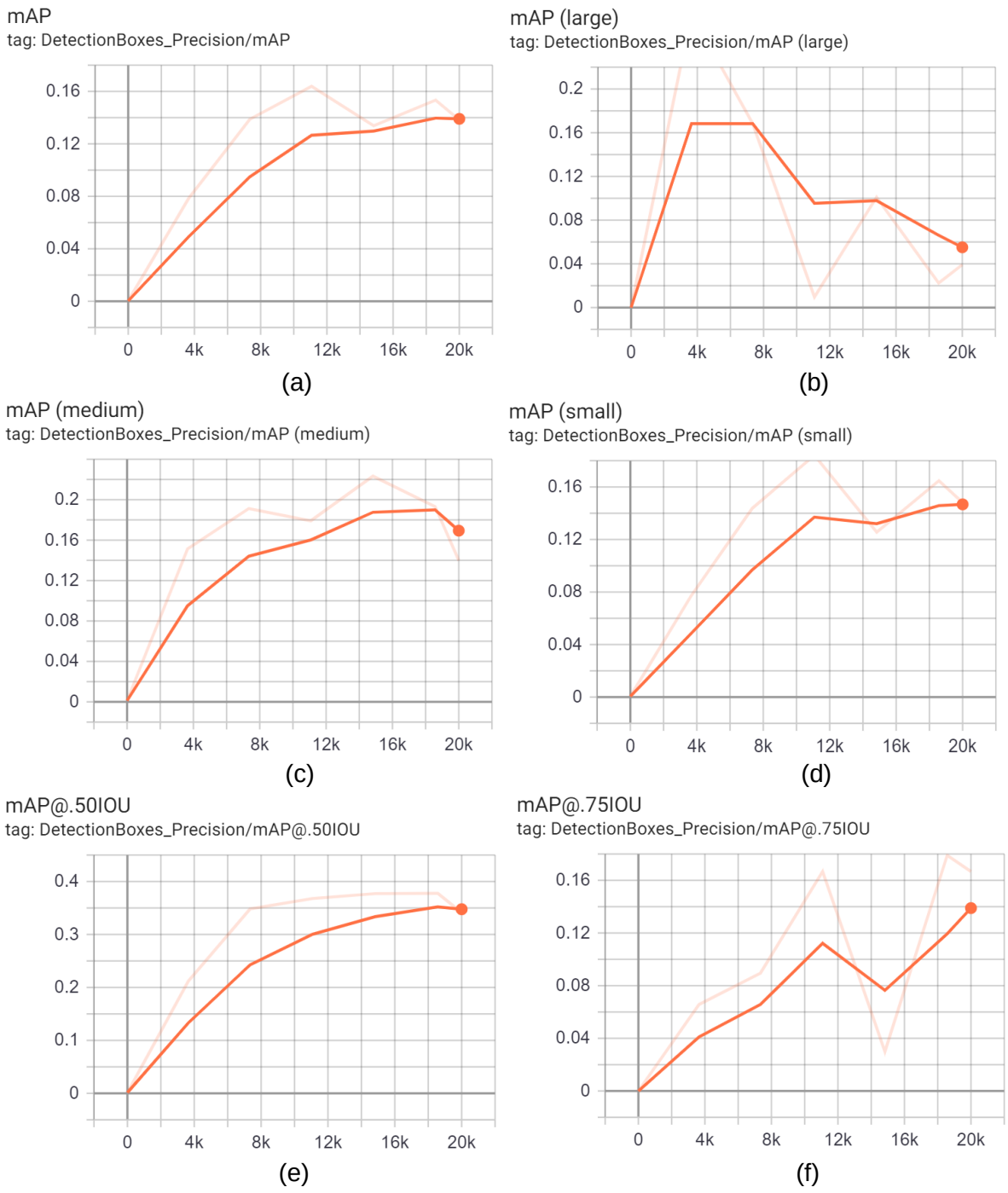


Figure 4-24 The results of 6 classes with 500 images at 20,000 step using mAP metric

4.1.2. Discussion of experiment 1

In this section, it will be divided into two main parts, which are the discussion of the detection results in images and analysis of the evaluation results from the COCO metric.

Firstly, the detection results in images of each model will be discussed. For one class experiment, overall, it can be seen that increasing the number of training image can improve the performance of the model. For example, figure 4-3 and figure 4-7 are the same images, but the model with 500 images predicted correctly, while the model with 20 images predicted cars as litter class. Even though most of trashes, including very small litter, were predicted correctly for the model trained by 500 images, but some objects such as boat was detected as litter. Thus, incorrectly predicted objects have been reviewed and annotated to their own classes. As a result, 6 classes have been identified.

For six classes models, the model of 20 training images seems to perform well to detect building, vehicle, and boat correctly. This model was also able to predict person accurately, even the numbers of person that labelled are tiny compare to other class, see table 3-3. However, it still missed classifying a lot of litter that appears in pictures. It can be noticed that an increase of the numbers of images allowed the models could detect litter better, while the ability to detect other class would reduce. For example, the model with small training images more accurate than the model with 500 images for boat and vehicle classes, shown in figure 4-11, 4-15, and 4-18.

In actual, when the number of training samples is increased, the performance should be increased as well. In this case, it might because the amount of training steps is not enough. Increasing the number of training iterations may help. Another possible problem is the numbers of labelled objects of each class are imbalance. Table 3-3 demonstrates that there are up to 1,400 litter that was labelled, while the amounts of labelled data of other classes are relatively small. This could make the model prejudice.

Secondly, the results in the COCO metric format will be visualised and assessed. The comparison of 6 models by each mAP metrics for 20,000 iterations illustrates in figure 4-25. Overall, it can be seen that the model 6 performed better than other models with around 0.15 of mAP, because mAP would compute the average of 10 IoU thresholds, i.e. 0.5, 0.55, …,0.95. Moreover, considering small objects in the data, model 6 performed outstandingly with mAP around 0.15, since there are much more small objects in training data for model 6. Besides, the results show that model 6 obtained high values for mAP@0.5IoU and mAP@0.75IoU. However,

there is a problem for large scale objects because this dataset consists of a small number of large objects since it is aerial images. This makes the training data lack large-scale data and makes each model difficult to predict large sizes of objects.

Model 3 that training with only one class with 1,424 labelled training of litter, see table 3-3, also achieved higher mAP metrics for mAP, mAP (medium), mAP(small), and mAP@.5IoU.

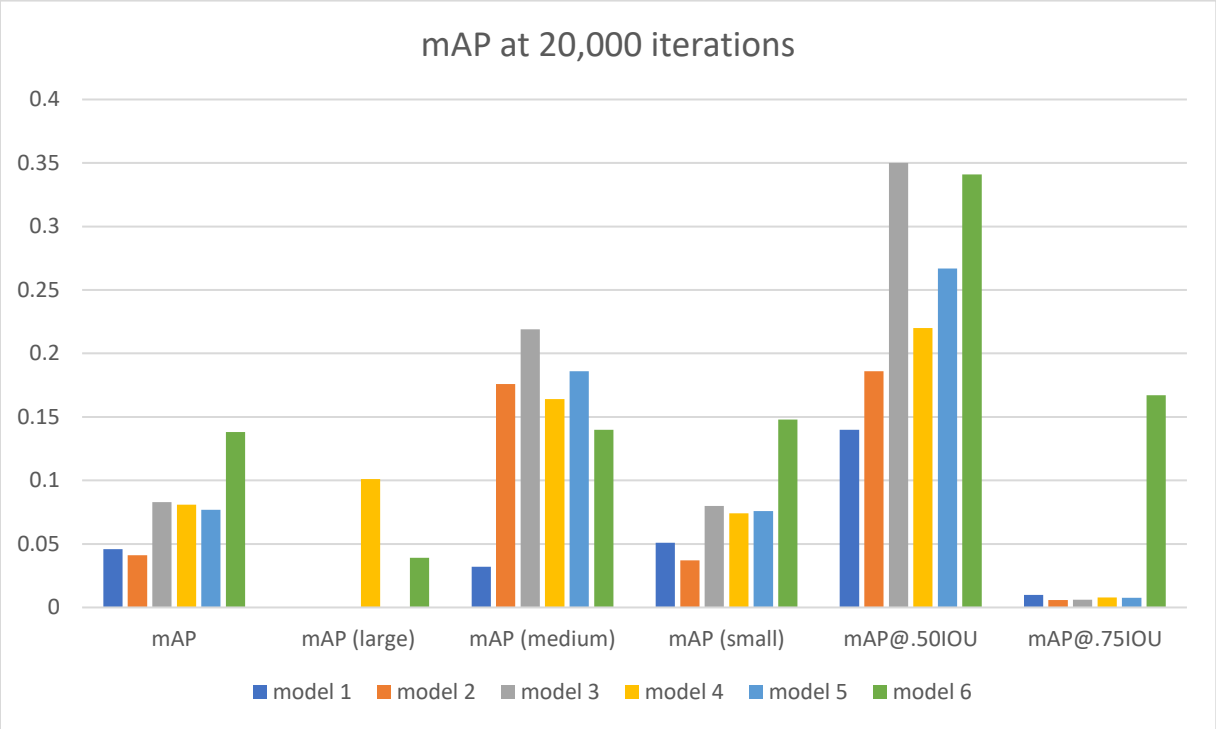


Figure 4-25 Comparison of 6 different models for each mAP score for 20,000 training step

Note: model 1 training with one class of litter using 20 training images

model 2 training with one class of litter using 100 training images

model 3 training with one class of litter using 500 training images

model 4 training with sixes class of litter using 20 training images

model 5 training with sixes class of litter using 100 training images

model 6 training with sixes class of litter using 500 training images.

It could be summarised that increasing the number of labelled training data would improve the performance of the object detection model for our dataset. In comparison, the model of 6

classes could perform better than one class model. As a consequence, the model of 6 classes and training with 500 images outperformed other models. Nevertheless, the mAP values in figure 4-25 are still quite low, which 0.35 maximum of mAP. In order to solve this, adding more training steps could help to improve performance. Another method is to try a different pre-trained model using other feature extractors.

4.2. Experiment 2

From experiment 1, the outcome of 6 classes training with 500 images gave the highest mAP score. Thus, this will be used to train with Faster R-CNN inception v2, Faster R-CNN Inception Resnet v2 and Faster R-CNN Resnet 101 COCO. This experiment will be trained until the loss converges close to 0.1 or be stable.

In experiment 2, model 7 will be the model that training with Faster R-CNN inception v2, model 8 will be the model training with Faster R-CNN Resnet 101 COCO and the model 9 will be the model training with Faster R-CNN Inception Resnet v2, as shown in table 4-1.

Table 4-1 Models in Experiment 2

Model	Pre-trained model
Model 7	Faster R-CNN inception v2
Model 8	Faster R-CNN Resnet 101 COCO
Model 9	Faster R-CNN Inception Resnet v2

Besides, these three models will be used to predict the unseen data with 100 images that contain litter and no litter in the images and focus only litter class. It means that if there is litter appears in an image (even only one piece), so it will be labelled as litter image (1), but if there is no litter, it will be labelled as no-litter (0). Then, comparing by using the confusion metric to compute Recall, Precision, Accuracy, and F-measure.

4.2.1. Result of Experiment 2

For mAP charts, the X-axis describes the number of training steps and the Y-axis describes the mAP score. For loss charts, the X-axis still represents the number of training steps, but the Y-axis represents the loss value. Figure 4-26 demonstrates the results of model 7 that show the

results of mAP, mAP (large), mAP (small), mAP@.5IoU, and mAP@.75IoU. Figure 4-2 shows the loss of model 7 that the loss stopped convergence at around step 20,000.

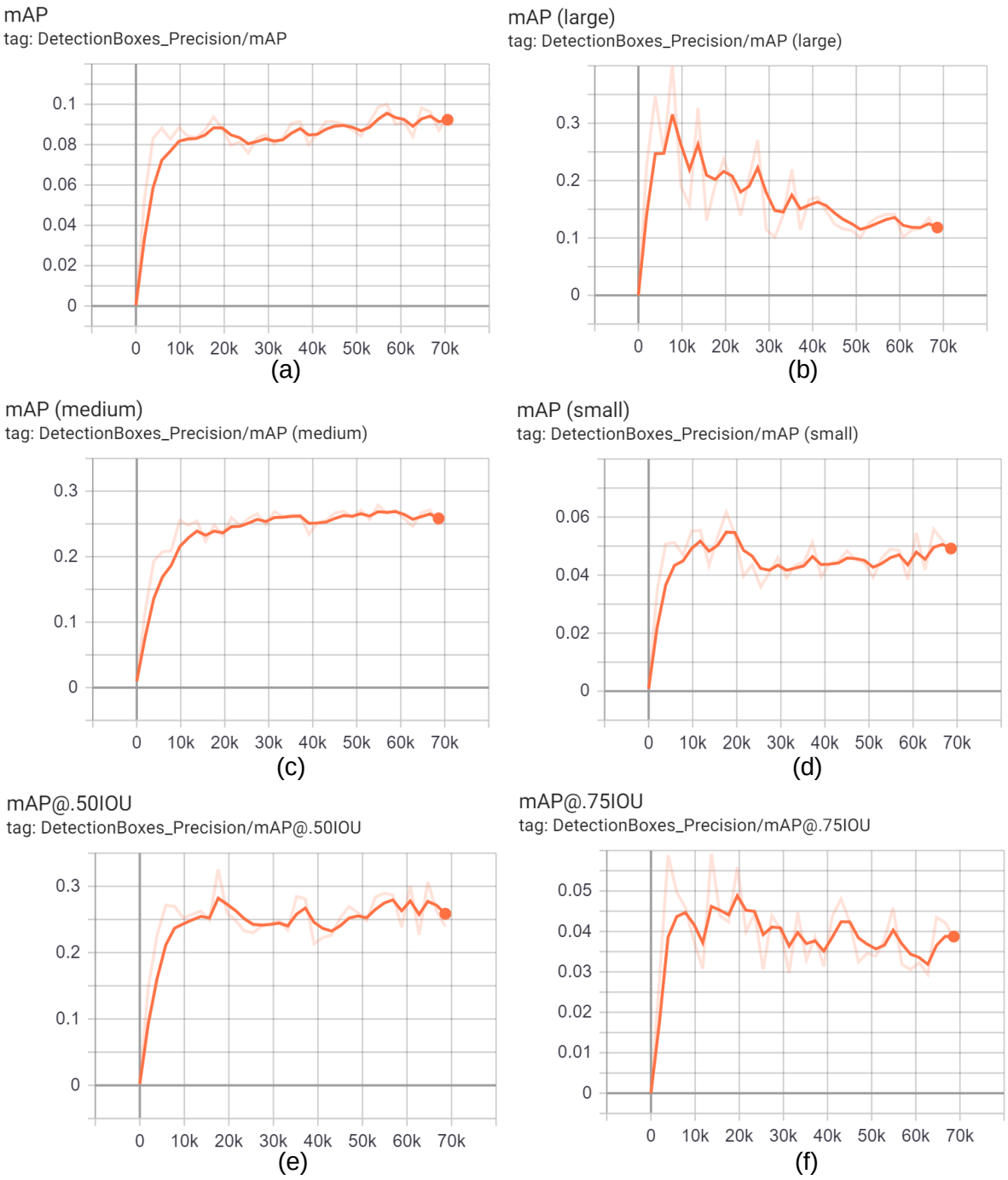


Figure 4-26 The mAP results of 6 classes with 500 images using Faster R-CNN Inception v2

Table 4-2 Loss of 6 classes with 500 images using Faster R-CNN Inception v2

The results that were generated from the pre-trained model Faster R-CNN Resnet 101 (model 8) and Faster R-CNN Inception Resnet (model 9) would provide a bit differ from the results that generated from Faster R-CNN Inception v2.

Figure 4-27 and 4-30 demonstrate the Average Precision at IoU threshold is 0.5 (AP@0.5IoU) of model 8 and model 9, respectively, in each category, i.e. boat, building, litter, person, tire, and vehicle. Figure 4-28 and 4-29 show the mAP score at IoU threshold is 0.5 (mAP@0.5IoU), and the loss value of model 8, reciprocally. For model 8, the loss seems stopped decrease since 10,000 steps, but it went down again at around 50,000 training steps. While mAP@0.5IoU jumped from step 0 to 8,000 and it rose slightly to approximately 0.32 at 70,000 iterations.

Figure 4-31 and 4-32 illustrate the mAP score at IoU threshold is 0.5 (mAP@0.5IoU) and the loss value of model 9, respectively. For model 9, the loss looks stable at around step 3,000, and mAP@0.5IoU increased gradually to 0.4 over about 70,000 training steps.

4.2.2. Discussion of Experiment 2

In experiment 2, the results of experiment 2 are quite challenging to assess because the format of the metric is not the same. However, every model still provided the mAP at 0.5 IoU, see figure 4-26 (e) for model 7, figure 4-28 for model 8, and 4-31 for model 9. The possible highest mAP@0.5IoU of model 7, model 8, and model 9 would be around 0.25, 0.36, and 0.4, respectively. It can be noticed that performance of model 8 and model 9 outperformed model 7. Thus, This will focus the comparison between model 8 and model 9, which were trained from Faster R-CNN Resnet 101 and Faster R-CNN Inception v2, reciprocally.

Overall, it is obvious that the detection precision of boat, building, litter, person, tire, and vehicle are improved when the number of training step increase.

Model 8, which trained from Faster R-CNN Resnet 101, was trained around 70,000 steps. It shows that Average Precision of building and vehicle class is better than Average Precision of boat, building, litter, and person, see figure 4-27. The reason might be small objects such as litter, person, boat, and tire are very difficult to detect because of their low resolution.

While the model 9, which trained from Faster R-CNN Inception Resnet v2, was trained only 20,000 iterations since figure 4-30 shows that Average Precision is stable when the training step is around 18,000. Moreover, this model took about 2.5 seconds per each training step. This is a problem when training on Google Colab because it has a limitation of the GPU usage and time limit for each notebook. Over 20,000 steps, the results show that Average Precision of tire is the highest score. It might be because there is only one shape for a tire. The second highest detection precision is building because the building is a large scale object, so it is easy to detect. While litter class also obtained high Average Precision score at 0.4.

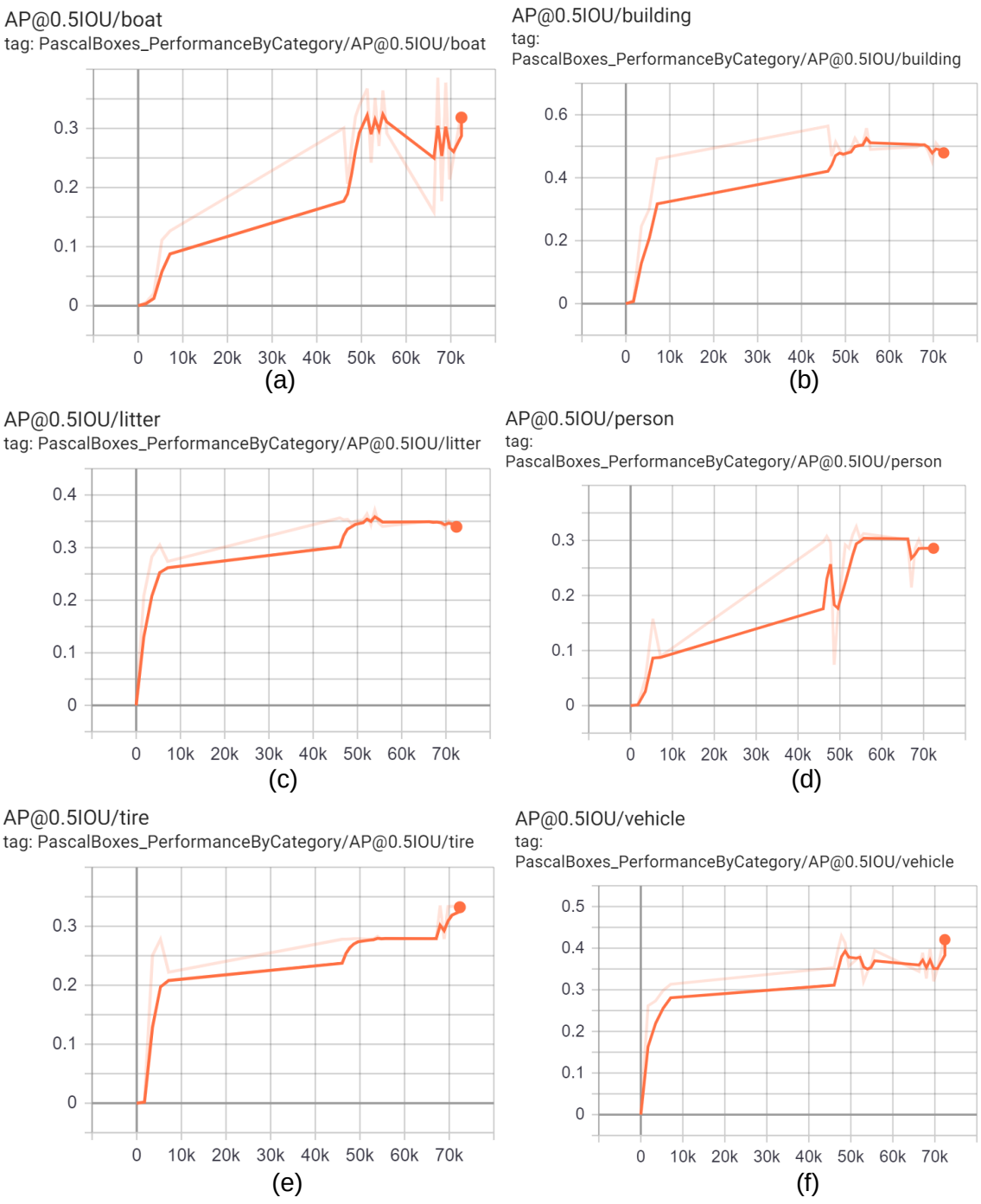


Figure 4-27 The AP results of 6 classes with 500 images using Faster R-CNN Resnet 101

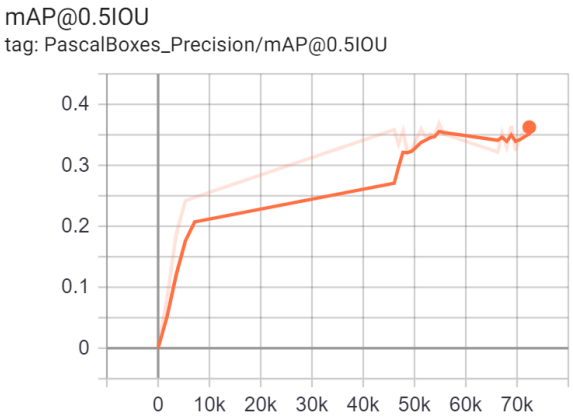


Figure 4-28 The mAP score at threshold 0.5 using Faster R-CNN Resnet 101

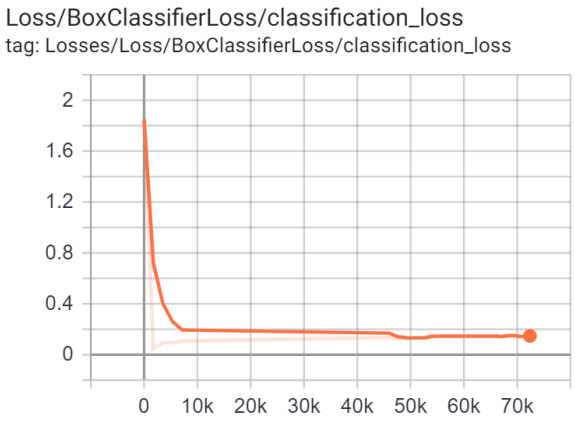


Figure 4-29 Loss of 6 classes with 500 images using Faster R-CNN Resnet 101

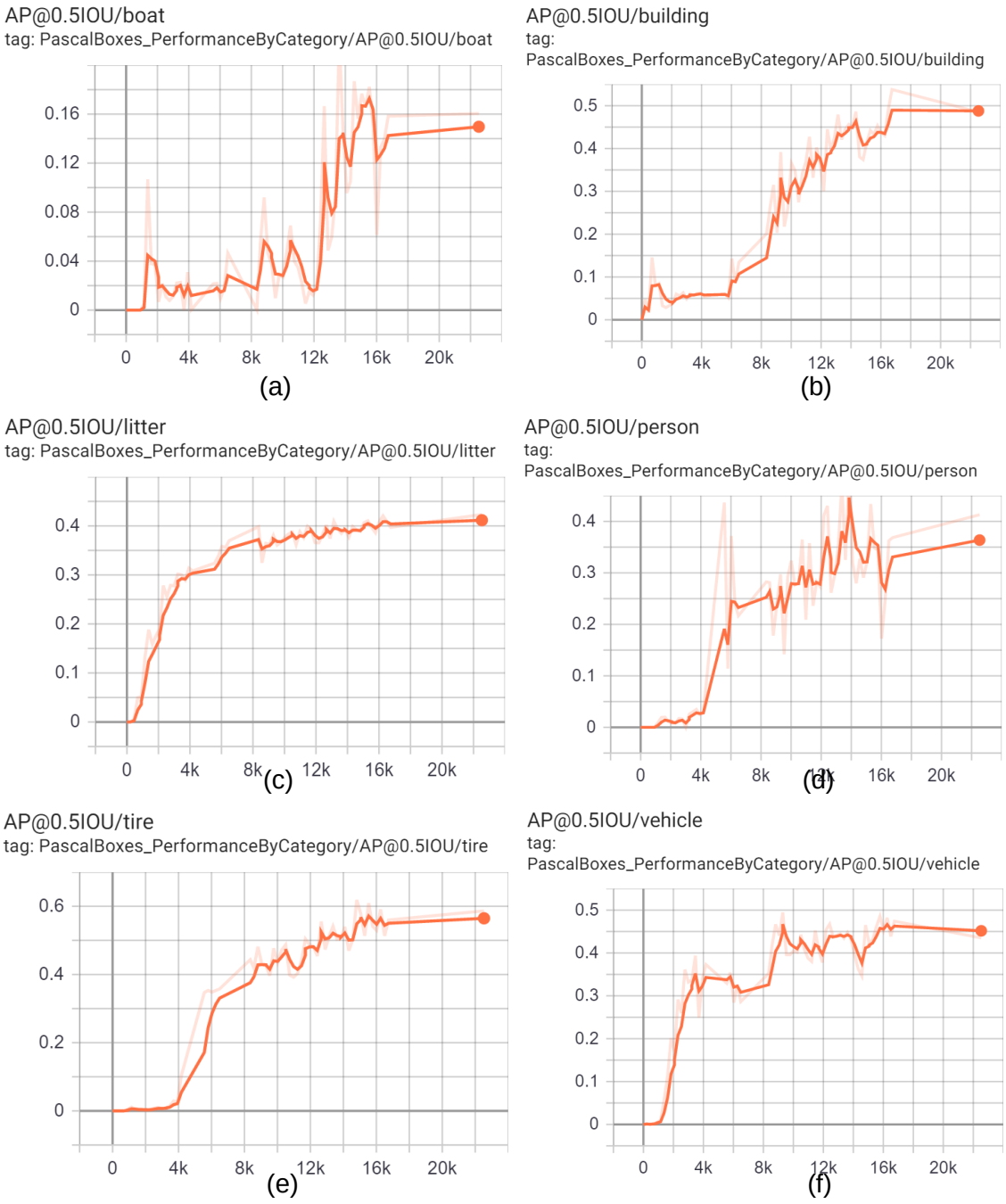


Figure 4-30 The AP results of 6 classes with 500 images using Faster R-CNN Inception Resnet

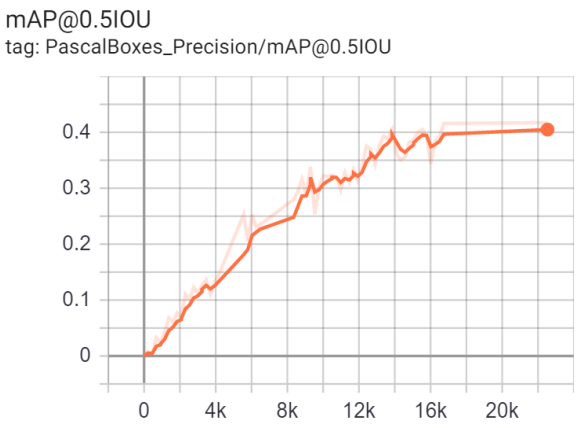


Figure 4-31 The mAP score at threshold 0.5 using Faster R-CNN Inception Resnet v2

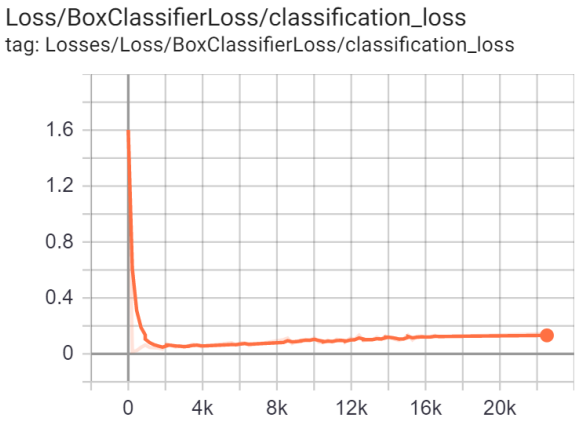


Figure 4-32 Loss of 6 classes with 500 images using Faster R-CNN Inception Resnet v2

Turning to the comparison of each detector from the sample detected results. Figure 4-33, 4-34, and 4-35 demonstrate predicted results of experiment 2 where image (a) of each figure was predicted from model 7, images (b) came from model 8, and images (c) came from model 9.

Figure 4-33 shows the image that contains a lot of cars. However, the prediction from model 7 in figure 4-33 (a), and model 8 in figure 4-33 (b) made incorrect predictions that predicted numerous cars as litter. While the prediction from model 9 in figure 4-33 (c) did not predict car as litter even it cannot detect any vehicle.

Focusing on litter detection, figure 4-34 and 4-35 illustrate examples of detection results of each model. From the outcomes, it is evident that the model 8 and model 9 could detect more litter than model 7, but they still missing some litter in some images.

Finally, these three models were used to detect testing data with 100 unseen images. However, only litter class was focused because we would like to know the performance of the model to

detect litter. Raw detection results can be seen in Appendix C. Table 4-3 presents the confusion matrix that can be used to compute Recall, Precision, Accuracy and F-measure of each model. Prediction results of the testing data show that model 9 obtained the highest score for Precision, Accuracy, and F-measure but model 8 got the highest recall. It could be seen that the overall performance of the model 9 is better than model 7 and model 8. Nevertheless, in this case, model 8 also provided the high efficiency for this task since it was the most accurate prediction of litter and the least predicted no litter that actual label is litter. It would not be suitable for this task to predict that there is no litter, but in fact, there is litter. In contrast, it would not be so bad if the model predicts as litter, but in reality, there is no litter because the user is able to observe the individual image again in case that image is classified as litter class.

To sum up, model 8 and model 9, which were trained from Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2 outperformed the model 7 that trained from Faster R-CNN Inception v2. However, it is difficult to summarise that which one is better between Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2 architectures. As a result, models were trained by Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2 provided an excellent performance that can be used in a real situation and help the user detect litter from the images automatically.

Table 4-3 The predicted results on testing data of each model

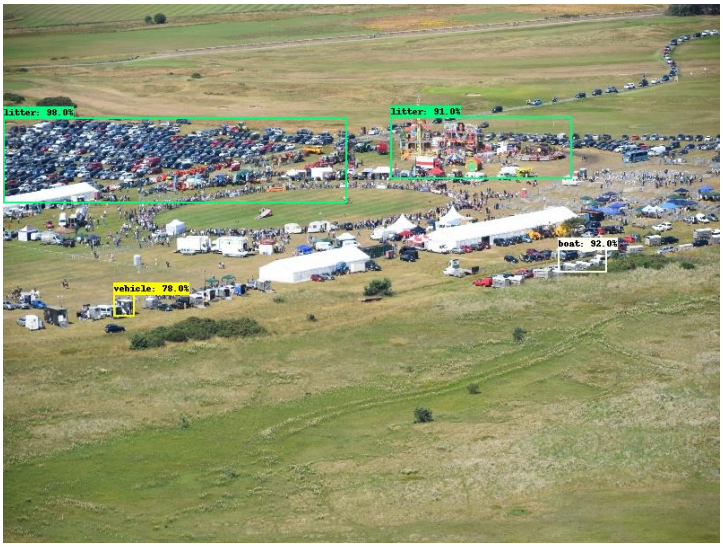
Model / Error	Confusion Matrix		Recall	Precision	Accuracy	F-measure
Model 7	33	15	0.90	0.76	0.80	0.82
	5	47				
Model 8	34	14	0.94	0.77	0.83	0.85
	3	49				
Model 9	39	9	0.92	0.84	0.87	0.88
	4	48				

The detail of the confusion matrix can be seen in table 4-4, more detail in section 3.6.1.

Table 4-4 The confusion matrix format

	Predicted: no litter	Predicted: litter
Actual: no litter	TN	FP
Actual: litter	FN	TP

(a)



(b)

(c)

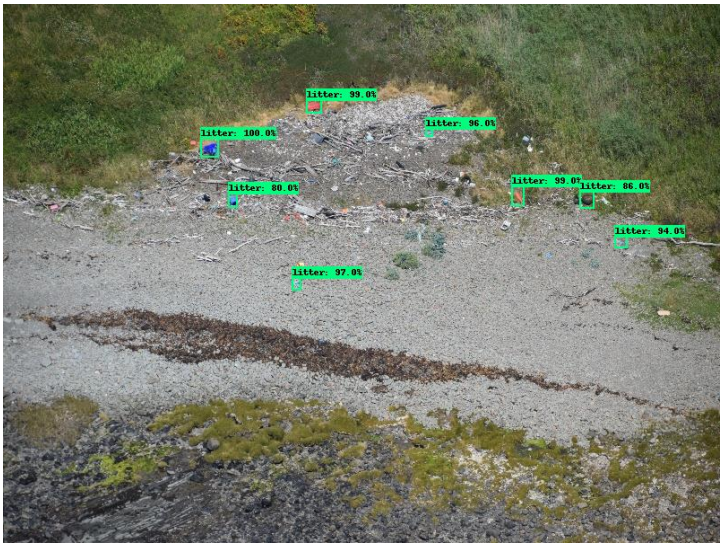
Figure 4-33 Detection results of model 7 (a), model 8 (b) and model 9 (c)

(a)

(b)

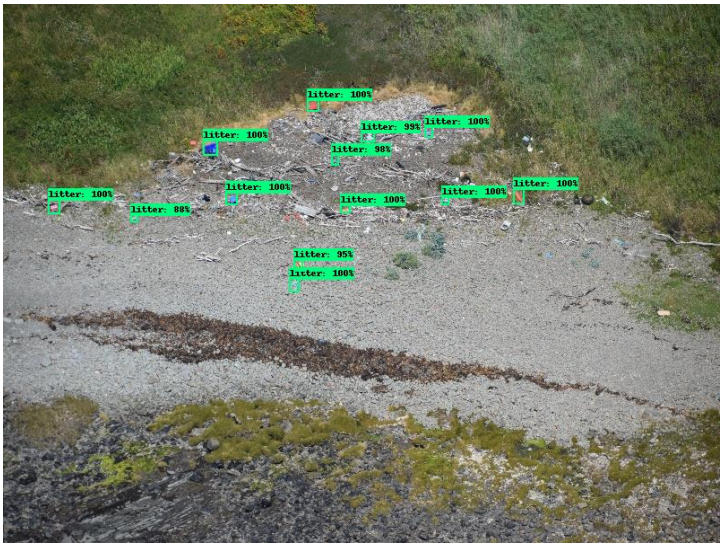
(c)

Figure 4-34 Detection results of model 7 (a), model 8 (b) and model 9 (c)



(a)

(b)



(c)

Figure 4-35 Detection results of model 7 (a), model 8 (b) and model 9 (c)

Chapter 5 Conclusion

The Scottish Coastal Rubbish Aerial Photography or SCRAPbook is an invention project that has been developed to save the ocean since there are many tons of litter that are discharged in the sea each year. Litter has been deposited in every environment that most local governments and international organisations struggle to address this issue. It has many effects on biodiversity, especially in marine life, including physical, biological, and chemical impacts. For example, marine creatures could misunderstand that plastic is their food.

Therefore, this project then presents a deep learning-based automatic litter detecting system that allows user would save a vast amount of time as they are able to use this system to detect aerial images instead of a human. The system could be built by using the popular technique of computer vision called object detection. The numbers of models were trained by Faster R-CNN architecture using a different type of feature extractor, such as Inception v2, Resnet 101, and Inception Resnet v2.

In this project, the experiment was divided into 2 parts. Firstly, research 1 will investigate how the performance changes if the number of labelled images and classes are increased by using Faster R-CNN inception v2 and 20000 training steps, which would be enough to make the loss converge close to zero. Then, experiment 2, the project will examine the appropriate model for our dataset by training the model with other pre-trained models that are Faster R-CNN Inception Resnet v2 and Faster R-CNN Resnet 101 COCO, which performs outstanding for small object detection. Finally, these three models were tested the accuracy using Recall, Precision, Accuracy, and F-measure.

The results from experiment 1 could be summarised that it would increase the efficiency of the object detection model by increasing the number of labelled training data. By contrast, the 6-class model could do better than a single class model. As a result, the 6-class model and 500-image training outperformed other models. However, experiment 1 provided low scores in mAP metric, thus different pre-trained model that using other feature extractors would have experimented. The outcomes from Experiment 2 could be concluded that models that trained by Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2 outperformed the model that trained from Faster R-CNN Inception v2. The results from the testing data also demonstrated high accuracy for Faster R-CNN Resnet101 and Faster R-CNN Inception Resnet v2. Nonetheless, Faster R-CNN Inception Resnet v2 would take much more training time than others.

Overall, the object detection model using deep learning for litter detection demonstrate pretty good performance that can help the user detect litter in the images automatically, even there is some error occur in the detection. The user would be able to use this system to classify that which image contains litter in the beginning. Then they could observe in an individual image that there is a possibility to contain litter.

During our experiments, other challenges have been found for litter detection model. Although this project is able to detect with 6-classes of objects, it would be interesting to annotate more free-form objects and broaden the number of class of litter. For example, the litter category could be separated into “litter made from plastic”, “litter made from the rubble”, etc. The future work would require more annotation objects and more data that can build the model to detect multi-scale items. Besides, it might focus on other pre-trained models that more suitable for our data.

Chapter 6 References

[1] K. Moy *et al.*, "Mapping coastal marine debris using aerial imagery and spatial analysis," *Marine Pollution Bulletin,* vol. 132, pp. 52-59, 2018, doi: 10.1016/j.marpolbul.2017.11.045.

[2] WWF. "What do sea turtles eat Unfortunately, plastic bags." https://www.worldwildlife.org/stories/what-do-sea-turtles-eat-unfortunately-plastic-bags (accessed June, 05, 2020).

[3] L. M. Rios, C. Moore, and P. R. Jones, "Persistent organic pollutants carried by synthetic polymers in the ocean environment," *Marine Pollution Bulletin,* vol. 54, no. 8, pp. 1230-1237, 2007, doi: 10.1016/j.marpolbul.2007.03.022.

[4] O. Conservancy. "Trash Free Seas Alliance." https://oceanconservancy.org/ (accessed June, 05, 2020).

[5] C. Ferries. "Ultimate Roundup of Marine Pollution Facts: The Causes and Impact on both Marine and Human Life." https://www.condorferries.co.uk/marine-ocean-pollution-statistics-facts#:~:text=There%20are%205.25%20trillion%20pieces,discarded%20in%20the%20sea%20yearly. (accessed June, 05, 2020).

[6] SCRAPbook. "SCRAPbook: Targeting Scotland's Coastal Litter." https://www.scrapbook-scotland.org.uk/ (accessed June, 05, 2020).

[7] SCRAPbook, "SCRAPbook Marine Litter Officers Case Study," ed, 2020.

[8] R. Shaoqing, H. Kaiming, R. Girshick, and S. Jian, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 6, pp. 1137-1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[9] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014: Springer, pp. 740-755.

[10] "Marine Scotland - A Marine Litter Strategy for Scotland," ed, 2014.

[11] M. Stachowitsch, *The Beachcomber's Guide to Marine Debris [internet resource]*, 1st ed. 2019.. ed. Cham : Springer International Publishing, 2019.

[12] S. Khan, *A guide to convolutional neural networks for computer vision / [internet resource]*. San Rafael, California : Morgan & Claypool, 2018.

[13] V. Wiley and T. Lucas, "Computer vision and image processing: a paper review," *International Journal of Artificial Intelligence Research,* vol. 2, pp. 29-36, 2018.

[14] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," ed, 1998, pp. 555-562.

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," vol. 1, ed, 2005, pp. 886-893 vol. 1.

[16] D. G. Lowe, "Object recognition from local scale-invariant features," vol. 2, ed, 1999, pp. 1150-1157 vol.2.

[17] X. Jiang, A. Hadid, Y. Pang, E. Granger, and X. Feng, *Deep Learning in Object Detection and Recognition [internet resource]*, 1st edition 2019.. ed. Singapore : Springer Singapore, 2019.

[18] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," ed, 2012, pp. 3642-3649.

[19] M. Venkatesan. "Artificial Intelligence vs. Machine Learning vs. Deep Learning." Data Science Central. https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning (accessed 2 September, 2019).

[20] P. Ping, G. Xu, E. Kumala, and J. Gao, "Smart Street Litter Detection and Classification Based on Faster R-CNN and Edge Computing," *International Journal of Software Engineering and Knowledge Engineering,* vol. 30, no. 04, p. 537, 2020, doi: 10.1142/S0218194020400045.

[21] E. Alpaydin, *Introduction to machine learning*, Third edition.. ed. Cambridge, Massachusetts : MIT Press, 2014.

[22] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed.. ed. Sebastopol: O'Reilly Media, Incorporated, 2017.

[23] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM,* vol. 60, no. 6, pp. 84-90, 2017, doi: 10.1145/3065386.

[24] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge.(Article)," *International Journal of Computer Vision,* vol. 115, no. 3, p. 211, 2015, doi: 10.1007/s11263-015-0816-y.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," R. Girshick, Ed., ed, 2014, pp. 580-587.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," vol. 2016-, ed, 2016, pp. 779-788.

[27] R. Girshick, "Fast R-CNN," *arXiv.org,* 2015.

[28] H. Kaiming, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," vol. 2017-, ed, 2017, pp. 2980-2988.

[29] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO," vol. 1004, ed, 2018.

[30] R. Joseph and F. Ali, "YOLOv3: An Incremental Improvement," ed, 2018.

[31] L. Tsung-Yi, P. Goyal, R. Girshick, H. Kaiming, and P. Dollar, "Focal Loss for Dense Object Detection," ed: IEEE, 2017, pp. 2999-3007.

[32] K. G. Dixit, M. G. Chadaga, S. S. Savalgimath, G. R. Rakshith, and N. Kumar, "Evaluation and Evolution of Object Detection Techniques YOLO and R-CNN," *International Journal of Recent Technology and Engineering (IJRTE),* vol. 8, no. 2S3, 2019.

[33] N. Fiedler, M. Bestmann, and N. Hendrich, "ImageTagger: An Open Source Online Platform for Collaborative Image Labeling," in *RoboCup*, 2018.

[34] *LabelImg*. (2015). [Online]. Available: https://github.com/tzutalin/labelImg

[35] *TensorFlow Object Detection API*. (2020). [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection

[36] Intel, "Traffic Light Detection Using the TensorFlow* Object Detection API," ed, 2018.

[37] Tensorflow, "Tensorflow 1 Detection Model Zoo," ed.

[38] N. Tajbakhsh *et al.*, "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?," *IEEE Transactions on Medical Imaging,* vol. 35, no. 5, pp. 1299-1312, 2016, doi: 10.1109/TMI.2016.2535302.

[39] H. Jonathan *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ed. Honolulu: HI, 2017, pp. 3296-3297.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

[42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167,* 2015.

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818-2826.

[44] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[45] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861,* 2017.

[46] Nickzeng. "An Introduction to Evaluation Matrics for Object Detection." https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/ (accessed June, 05, 2020).

[47] T. Carneiro, R. V. M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. Reboucas Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access,* vol. 6, pp. 61677-61685, 2018.

Chapter 7 Appendices

Appendix A: Implementing Object Detection using TensorFlow API on Google Colab

The benefit of using Google Colab is it allows user can train a model on a cloud with a free GPU and free storage. Colab notebook for implementing the task can be seen in Appendix E and Appendix F.

7.1.1. Colab setting

a.) Change run-time type in Colab to be GPU

b.) Install a tensorflow version 1.x. This API still uses some functions from Tensorflow version 1.x, while there is no in version 2.x.

c.) Restart run-time

d.) Mount Google Drive that is going to use.

e.) Create the project folder in Google Drive.

f.) Change directory to our project folder

g.) Clone repository of Tensorflow API into our folder using

```
!git clone https://github.com/tensorflow/models.git
```

h.) Install Protocol Buffers and Cython

```
!apt-get install protobuf-compiler python-pil python-lxml
python-tk
!pip install Cython
```

7.1.2. Labelling data

We recommend using LabelImg annotation tool for annotate objects because it is easy to generate TFRecord file.

7.1.3. Creating the dataset for training (TFRecord)

A TFRecord is the file type of dataset that store images and position of bounding boxes in one file, which can be used for Tensorflow object detection API.

This step will be done on local machine using Python. Then, uploading them into google colab.

7.1.4. Configuring training

Before training the model, last thing would be done is to generate a label map and a training configuration file. A label map can be created in a local machine

and save as “file name”.pbtxt file. The pattern of the label map follows the pattern below:

```
item{
        id: 1
        name: 'litter'
}

item{
        id: 2
        name: 'vehicle'
}
item{
        id: 3
        name: 'building'
}
item{
        id: 4
        name: 'boat'
}
item{
        id: 5
        name: 'person'
}
item{
        id: 6
        name: 'tire'
}
```

The training configuration can be run a cell in this colab notebook.

7.1.5. Training the model (Colab notebook in Appendix E)

After uploading all files, it can be trained by run the training cell.

7.1.6. Exporting inference graph (Colab notebook in Appendix E)

This process can be used save the trained model and then use it later.

7.1.7. Evaluating the model (Colab notebook in Appendix E)

Checking the performance of the model.

7.1.8. Testing the model (Colab notebook in Appendix F)

Testing a model using unseen data.

Appendix B: The Results of Our Experiments

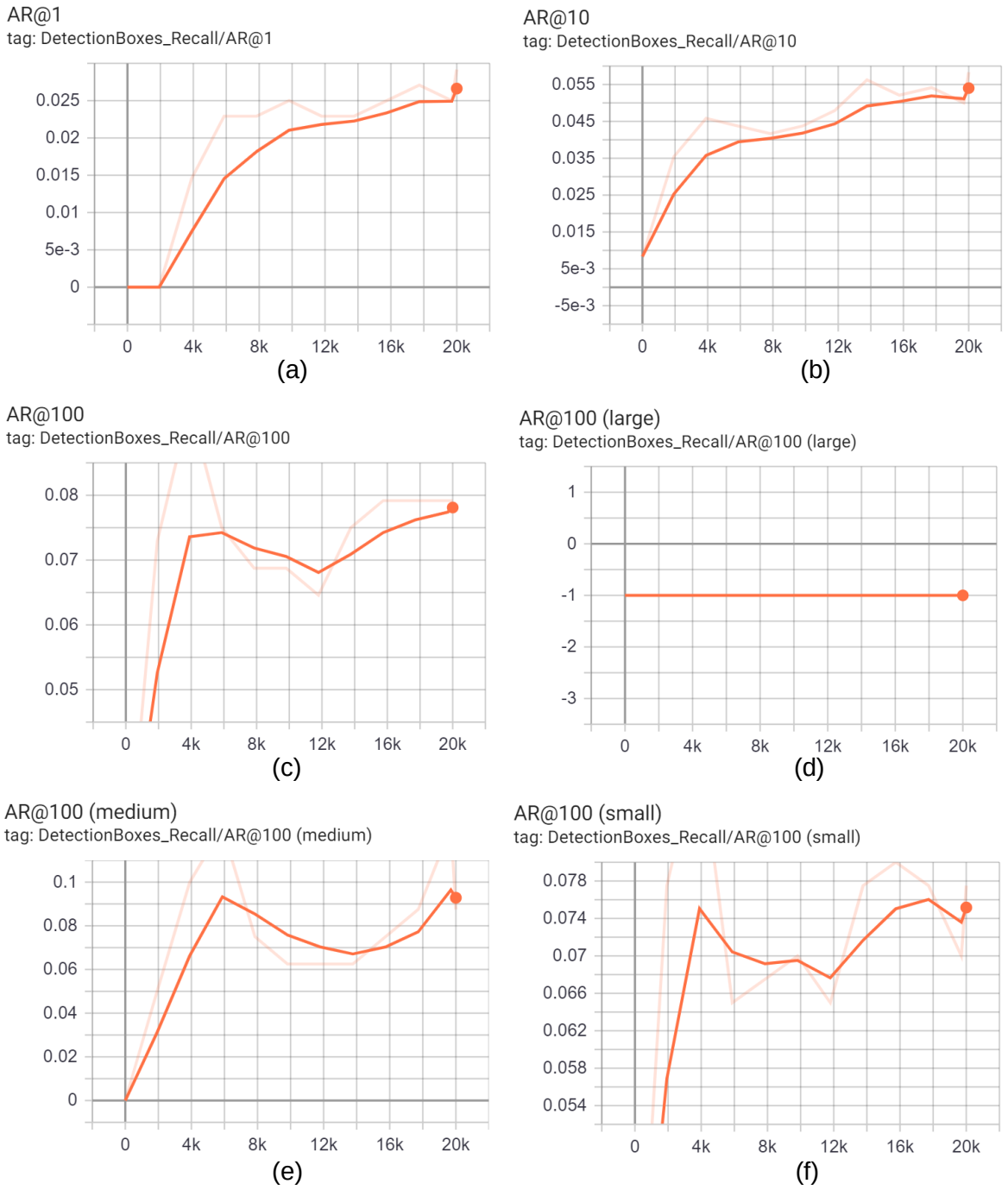


Figure 7-1 The results of 1 class with 20 images at 20,000 step using AR metric

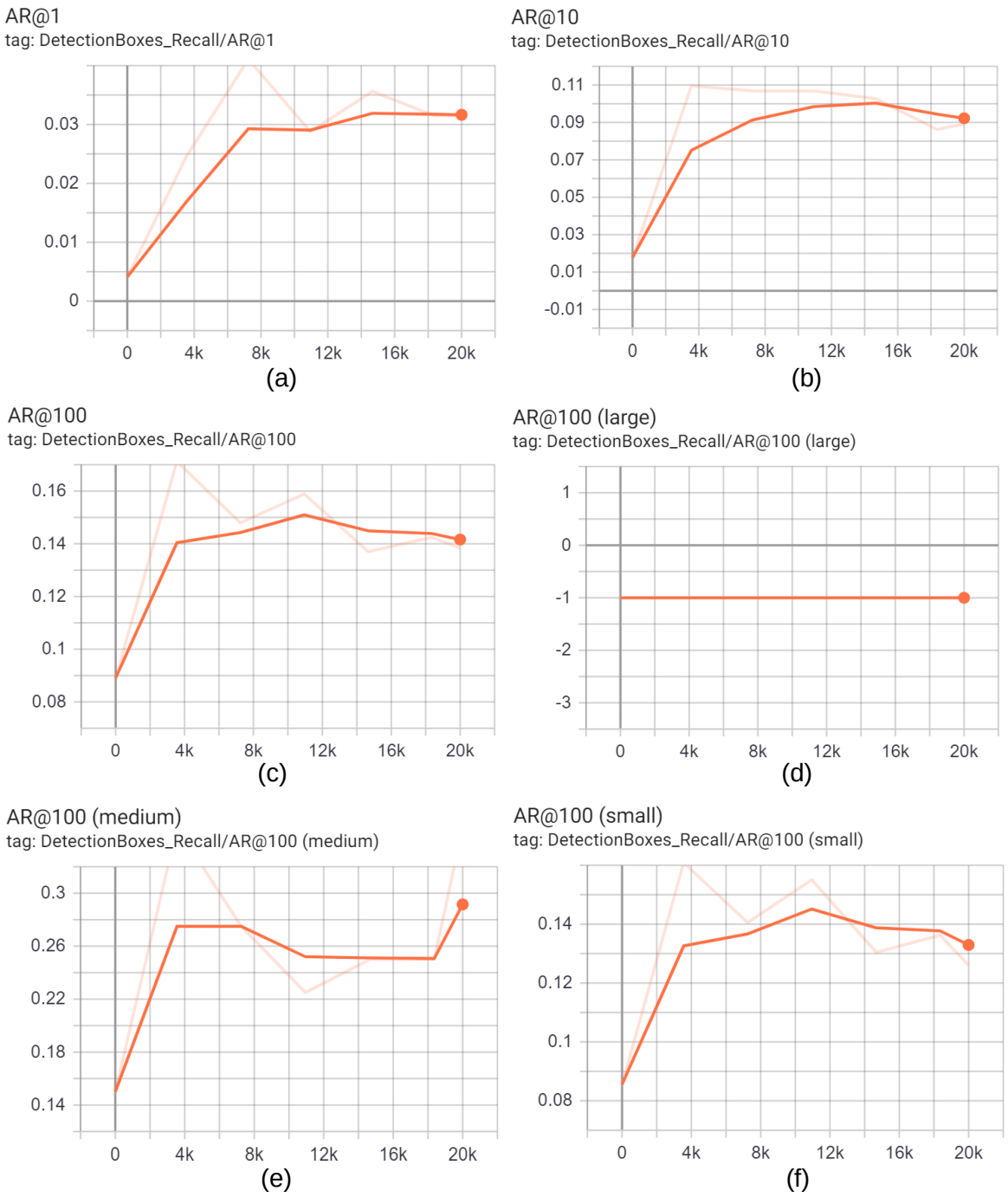


Figure 7-2 The results of 1 class with 100 images at 20,000 step using AR metric

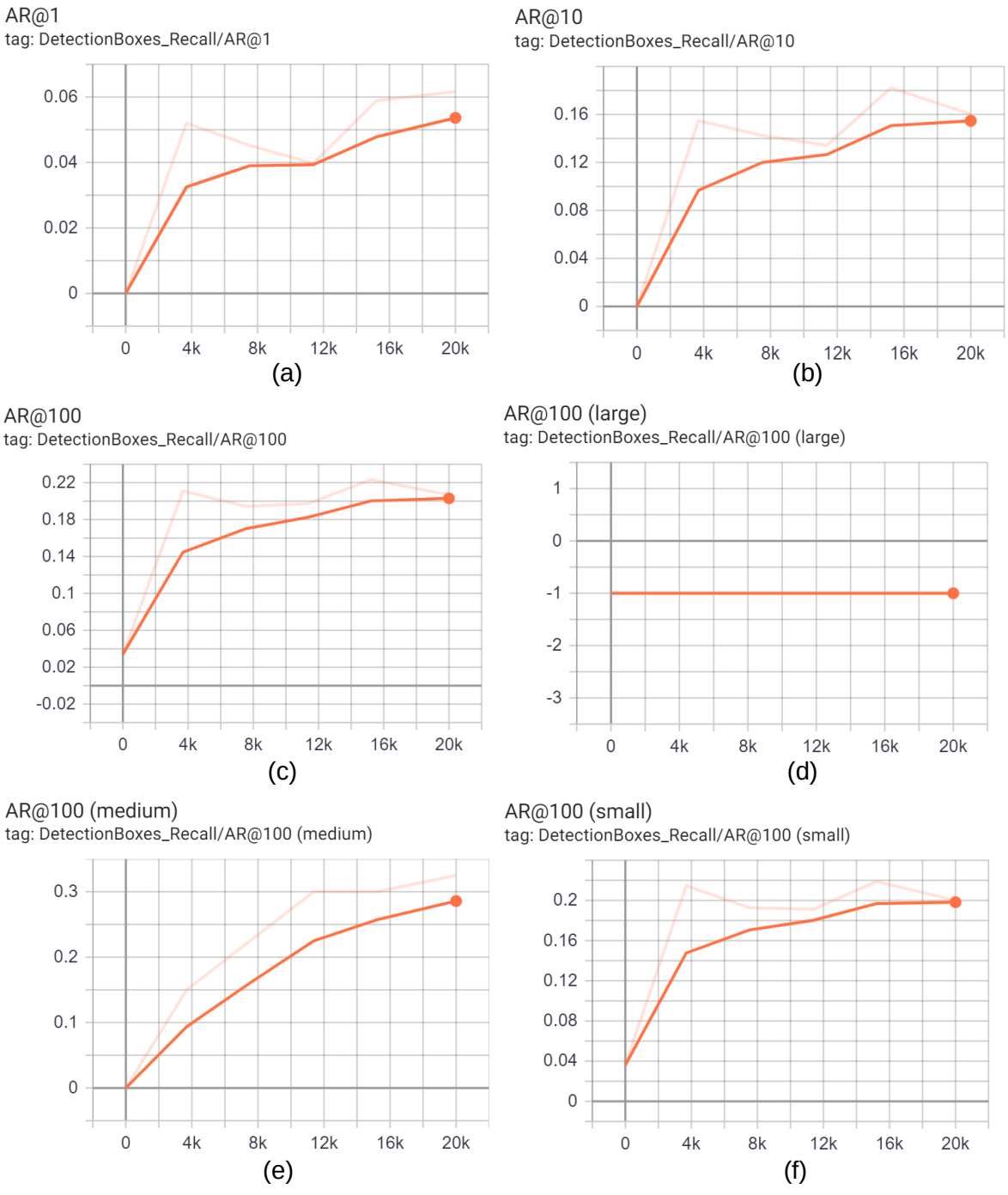


Figure 7-3 The results of 1 class with 500 images at 20,000 step using AR metric

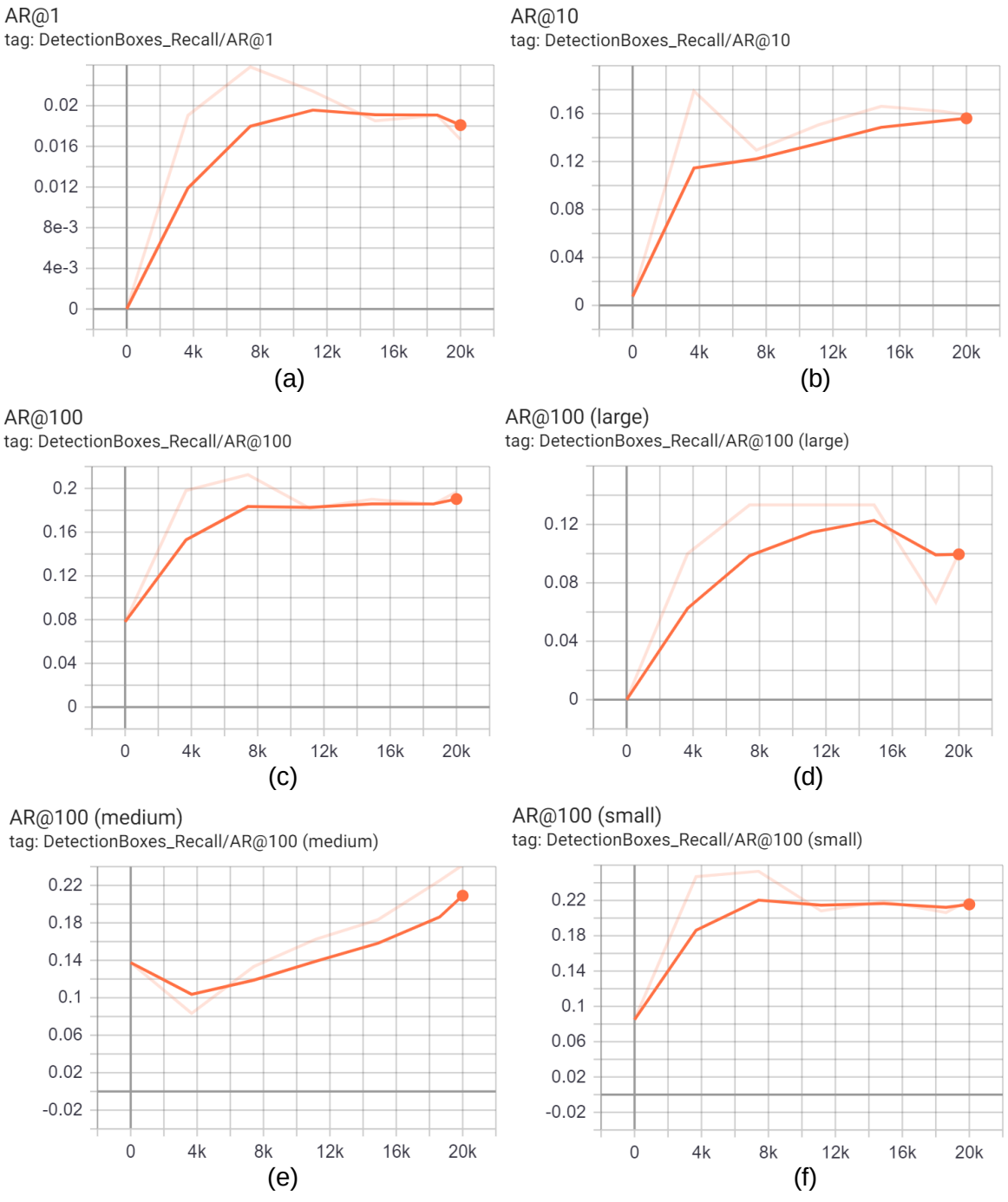


Figure 7-4 The results of 6 classes with 20 images at 20,000 step using AR metric

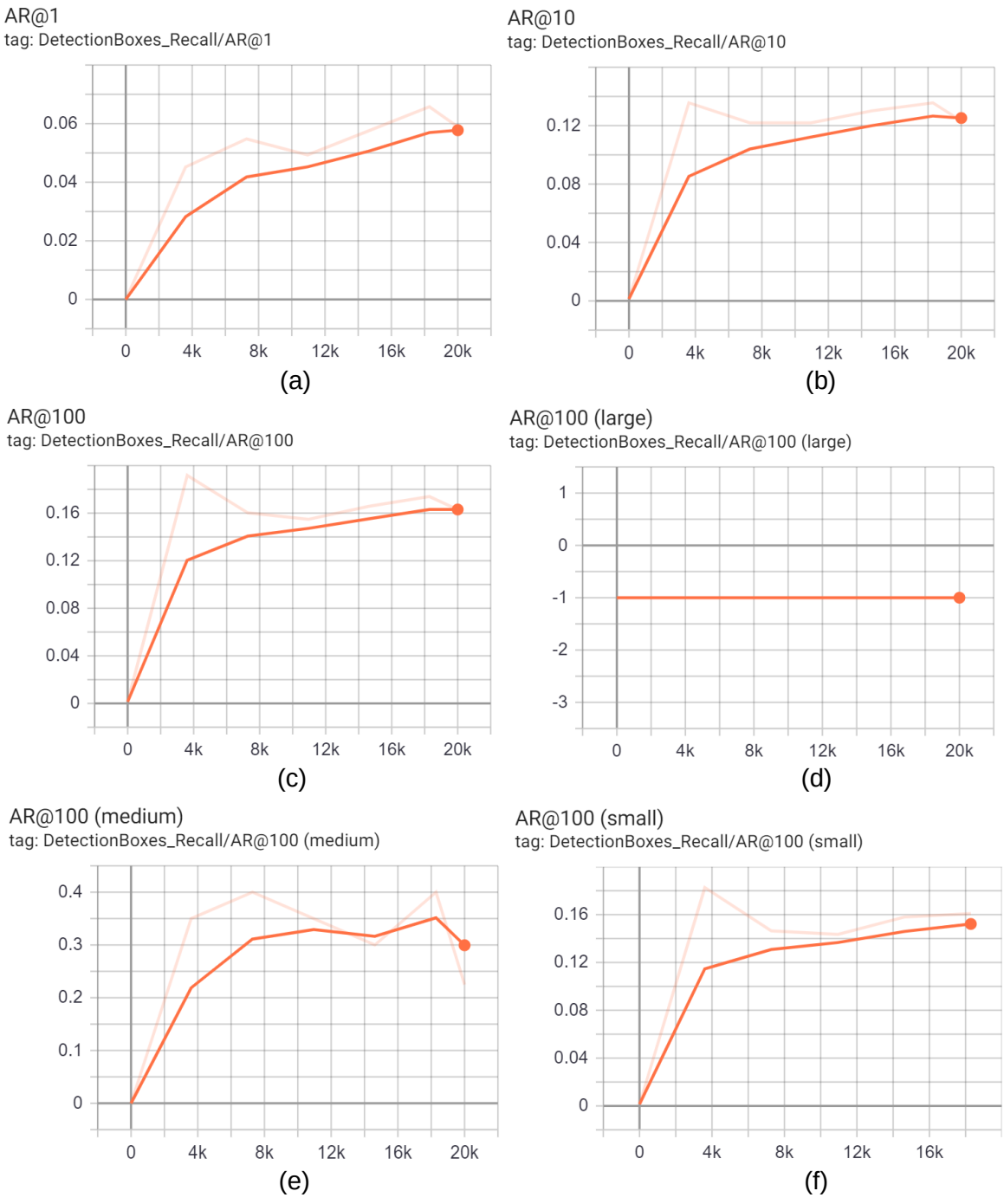


Figure 7-5 The results of 6 classes with 100 images at 20,000 step using AR metric

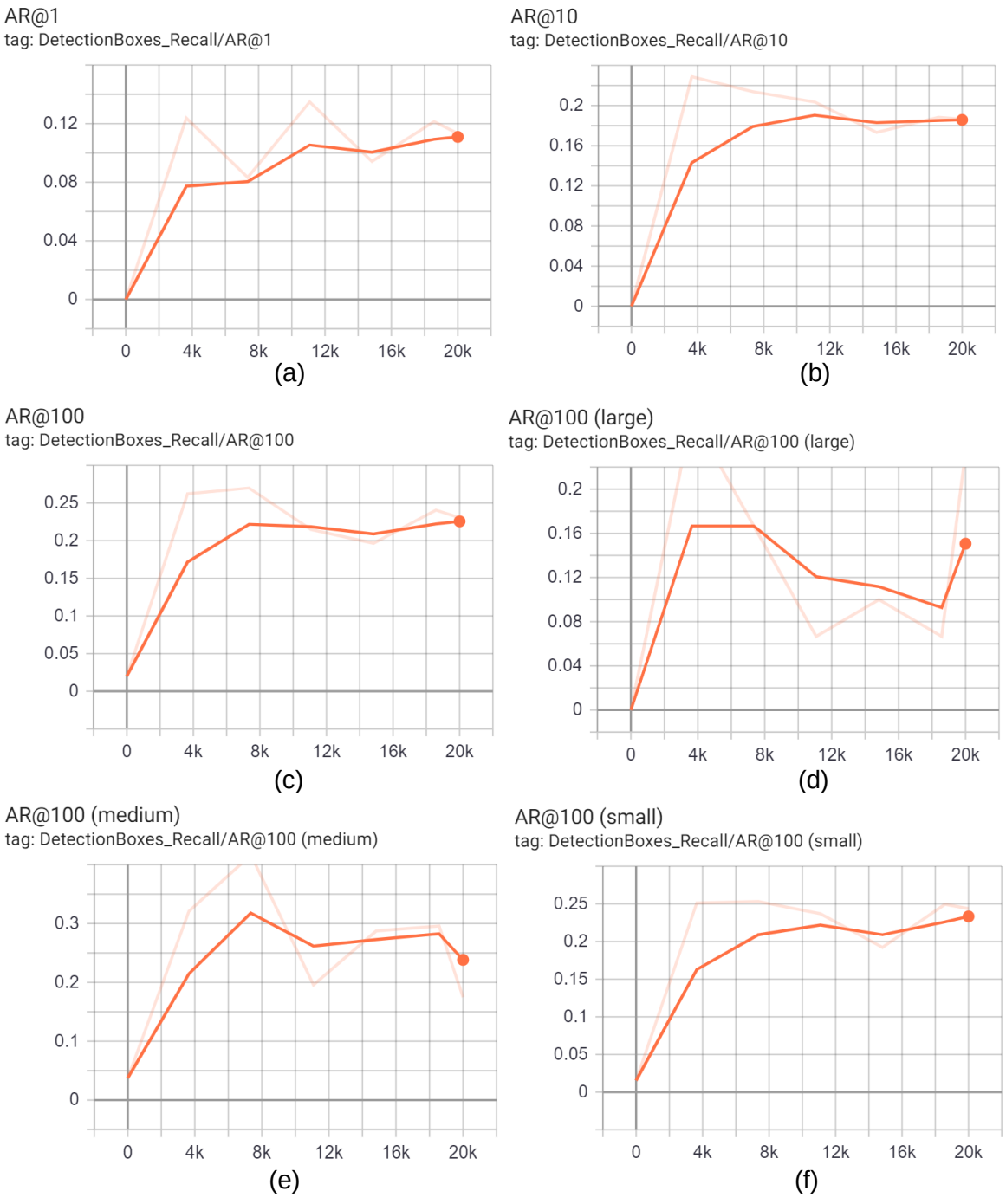


Figure 7-6 The results of 6 classes with 500 images at 20,000 step using AR metric

Figure 7-7 The AR results of 6 classes with 500 images using Faster R-CNN Inception at around 70,000 step

Appendix C: 100 Testing Data (Unseen Images)

Image 1.jpg to 6.jpg

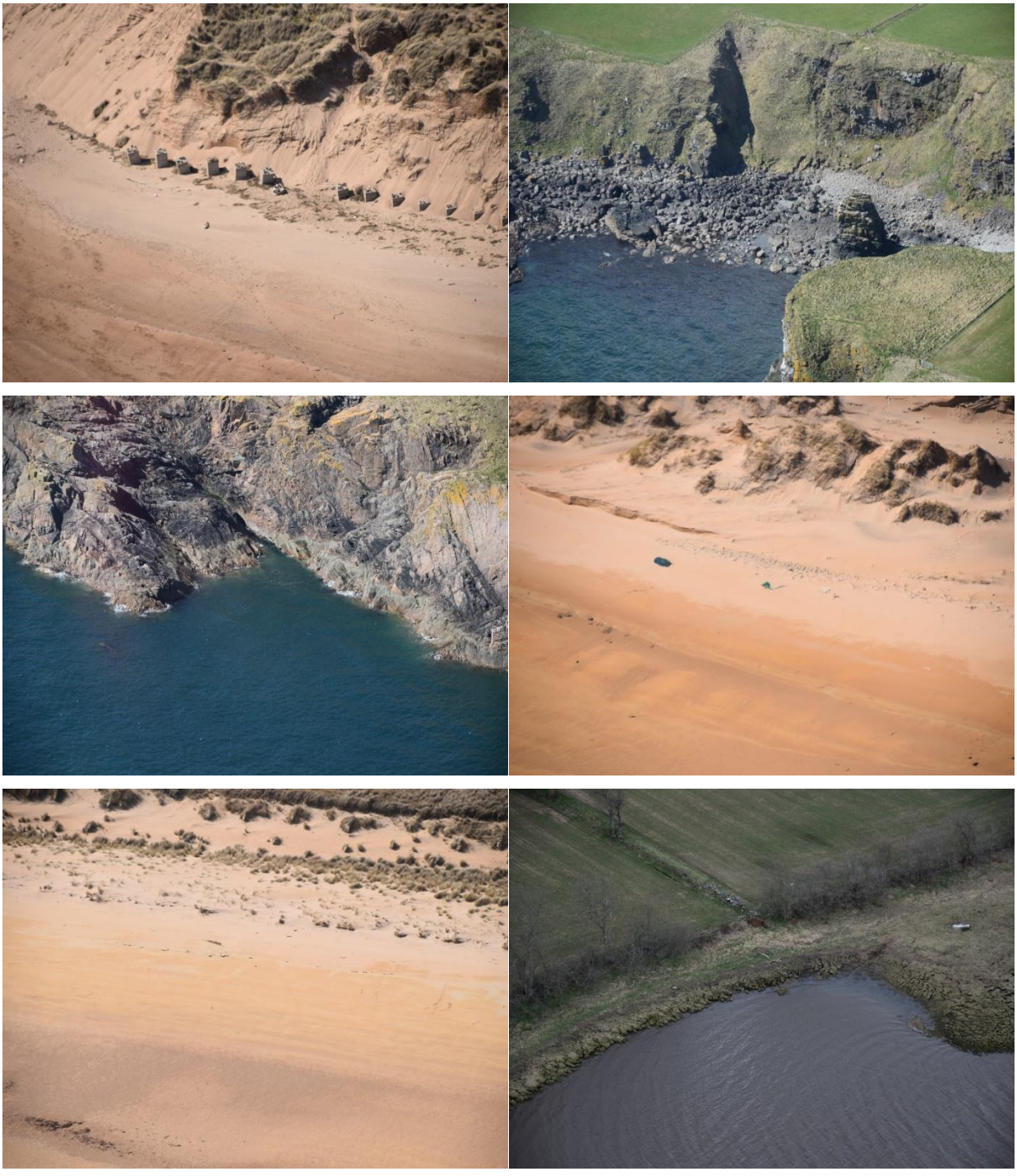


Image 7.jpg to 12.jpg

Image 13.jpg to 18.jpg

Image 19.jpg to 24.jpg

Image 25.jpg to 30.jpg

Image 31.jpg to 36.jpg

Image 37.jpg to 42.jpg

Image 43.jpg to 48.jpg

Image 49.jpg to 54.jpg

Image 55.jpg to 60.jpg

Image 61.jpg to 66.jpg

Image 67.jpg to 72.jpg

Image 73.jpg to 78.jpg

Image 79.jpg to 84.jpg

Image 85.jpg to 90.jpg

Image 91.jpg to 96.jpg

Image 97.jpg to 100.jpg

Appendix D: The Results of Detection from 100 Unseen Images

Table 7-1 The Results from each model and the actual label

image_name	y_actual	y_pred_inception_v2 (Model 7)	y_pred_resnet101 (Model 8)	y_pred_inception_resnet (Model9)
1.jpg	0	0	0	0
2.jpg	0	0	0	0
3.jpg	0	0	0	0
4.jpg	1	1	1	1
5.jpg	0	0	0	0
6.jpg	0	1	1	0
7.jpg	1	1	1	1
8.jpg	0	1	1	0
9.jpg	1	1	1	1
10.jpg	0	0	0	0
11.jpg	0	1	1	1
12.jpg	0	0	0	0
13.jpg	0	0	0	0
14.jpg	0	0	0	0
15.jpg	0	0	0	0
16.jpg	0	0	0	0
17.jpg	0	0	0	0
18.jpg	0	1	1	0
19.jpg	0	1	1	1
20.jpg	0	0	0	0
21.jpg	1	1	1	1
22.jpg	0	0	0	0
23.jpg	0	1	1	1
24.jpg	0	0	0	0
25.jpg	1	0	0	0
26.jpg	0	1	1	1
27.jpg	1	1	1	1

28.jpg	1	1	1	1
29.jpg	1	1	1	1
30.jpg	1	1	1	1
31.jpg	0	0	0	0
32.jpg	1	1	1	1
33.jpg	1	1	1	1
34.jpg	1	1	1	1
35.jpg	1	0	1	1
36.jpg	1	1	1	1
37.jpg	1	1	1	1
38.jpg	1	1	1	1
39.jpg	1	1	1	1
40.jpg	1	1	1	1
41.jpg	1	1	1	1
42.jpg	1	1	1	1
43.jpg	0	1	0	1
44.jpg	0	1	1	0
45.jpg	1	1	1	1
46.jpg	1	1	1	1
47.jpg	1	1	1	1
48.jpg	1	1	1	1
49.jpg	1	1	1	1
50.jpg	1	1	1	1
51.jpg	1	1	1	1
52.jpg	1	1	1	1
53.jpg	1	1	1	1
54.jpg	1	1	1	1
55.jpg	1	1	1	1
56.jpg	1	1	1	1
57.jpg	1	1	1	1
58.jpg	0	1	0	1

59.jpg	1	1	1	1
60.jpg	0	0	0	0
61.jpg	0	0	0	0
62.jpg	0	0	0	0
63.jpg	0	1	1	1
64.jpg	0	0	1	0
65.jpg	1	0	0	0
66.jpg	1	0	1	0
67.jpg	0	0	0	0
68.jpg	0	0	0	0
69.jpg	0	0	0	0
70.jpg	0	0	1	0
71.jpg	1	1	1	1
72.jpg	1	1	1	1
73.jpg	1	1	1	1
74.jpg	1	1	1	1
75.jpg	1	1	1	1
76.jpg	1	1	1	1
77.jpg	1	0	1	1
78.jpg	0	1	1	1
79.jpg	0	1	1	0
80.jpg	1	1	1	1
81.jpg	1	1	1	1
82.jpg	0	1	0	0
83.jpg	1	1	1	1
84.jpg	1	1	1	1
85.jpg	1	1	1	1
86.jpg	1	1	1	1
87.jpg	0	0	0	0
88.jpg	1	1	1	1
89.jpg	0	0	0	0

90.jpg	0	0	0	0
91.jpg	0	0	0	1
92.jpg	0	0	0	0
93.jpg	1	1	0	0
94.jpg	0	1	1	0
95.jpg	0	0	0	0
96.jpg	0	0	0	0
97.jpg	1	1	1	1
98.jpg	0	0	0	0
99.jpg	0	0	0	0
100.jpg	0	0	0	0

Table 7-2 The result of the prediction from model 7 (Trained with Faster R-CNN Inception v2)

image_name	class	ymin	xmin	ymax	xmax
11.jpg	litter	284.4596	309.3195	301.0035	350.5449
18.jpg	litter	174.125	293.0476	191.9058	304.4073
19.jpg	litter	39.21179	380.6036	52.10667	399.9588
19.jpg	litter	142.4019	645.114	160.8657	663.9518
19.jpg	litter	161.1833	498.8855	174.308	509.418
21.jpg	litter	258.4287	361.0975	268.2014	370.4543
21.jpg	litter	282.5472	372.9165	302.3256	389.7147
23.jpg	litter	375.5372	322.6565	430.7347	376.596
26.jpg	litter	234.2586	513.1345	241.6965	523.6125
27.jpg	litter	244.7807	220.8316	265.6529	247.7213
28.jpg	litter	384.0812	464.7306	394.8524	472.2208
29.jpg	litter	372.8058	454.0706	386.6465	467.497
30.jpg	litter	172.8479	428.2752	183.471	438.9317
32.jpg	litter	197.7839	295.1704	217.0987	315.2215
33.jpg	litter	233.0087	384.1362	244.0409	394.0595
33.jpg	litter	295.7491	410.1862	312.0439	430.9487

34.jpg	litter	266.914	414.7646	282.5944	435.9137
36.jpg	litter	11.62987	637.9758	27.1111	668.9535
36.jpg	litter	348.9883	367.423	361.8331	378.567
36.jpg	litter	342.7739	255.8736	353.3734	267.3546
37.jpg	litter	282.7156	342.5861	298.5877	354.7338
37.jpg	litter	311.1182	403.972	326.6333	419.6867
38.jpg	litter	318.1435	533.2872	328.6407	541.8209
39.jpg	litter	318.9428	353.7871	329.4006	366.0241
39.jpg	litter	270.8474	207.6523	289.1997	224.7541
39.jpg	litter	299.9803	209.1627	311.8241	221.5292
4.jpg	litter	252.231	227.4843	270.2896	249.3813
4.jpg	litter	291.6621	401.5127	307.6252	426.684
40.jpg	litter	348.0773	288.4043	366.7049	308.7478
40.jpg	litter	386.1395	348.4408	406.622	363.7733
41.jpg	litter	348.1931	498.3648	363.2026	515.4995
41.jpg	litter	360.654	108.8774	390.542	151.6251
41.jpg	litter	258.8064	385.4892	272.2256	398.4725
41.jpg	litter	254.8436	515.3883	267.7159	527.9773
41.jpg	litter	300.1929	698.8042	314.0401	708.4339
42.jpg	litter	284.9825	265.7296	295.9763	280.1489
42.jpg	litter	271.1451	370.5759	279.8598	379.3419
43.jpg	litter	355.7762	409.227	375.3125	422.9157
44.jpg	litter	127.1278	208.9064	190.9479	306.3062
45.jpg	litter	145.9265	166.5525	158.8811	175.6634
45.jpg	litter	116.3579	249.1925	130.084	265.1966
45.jpg	litter	276.0932	224.9563	287.4713	239.4058
45.jpg	litter	511.4728	492.4201	532.438	518.8667
45.jpg	litter	416.2883	641.9691	450.5193	673.4574
46.jpg	litter	285.3301	382.6332	313.3312	407.1013
46.jpg	litter	151.9537	238.0547	190.4994	266.1508
46.jpg	litter	249.053	457.7237	262.6409	466.7915

46.jpg	litter	187.0556	352.6978	200.1624	362.5503
46.jpg	litter	182.4011	407.8071	205.4519	433.6583
46.jpg	litter	190.092	790.2328	204.3442	798.9963
46.jpg	litter	220.9647	310.5666	229.205	322.2572
46.jpg	litter	250.5492	398.5319	267.5033	435.4198
47.jpg	litter	209.7504	322.5072	224.7233	340.9501
47.jpg	litter	98.27918	157.9263	109.2021	168.2803
48.jpg	litter	189.2116	454.9333	198.6016	463.0927
48.jpg	litter	236.1367	731.7968	255.3209	744.5511
48.jpg	litter	214.2339	275.9791	227.0762	286.7173
48.jpg	litter	203.3329	399.2763	214.0504	411.1623
49.jpg	litter	345.729	28.63851	364.1108	47.03266
49.jpg	litter	289.0015	402.6451	304.1553	412.459
49.jpg	litter	291.5607	300.9517	308.5969	315.5027
49.jpg	litter	254.7414	613.868	267.9282	625.2117
50.jpg	litter	260.5746	417.0949	277.3536	436.6293
50.jpg	litter	354.5326	288.7543	362.2902	298.2529
50.jpg	litter	257.772	608.9066	275.8778	624.6755
50.jpg	litter	214.3832	648.2203	224.9976	656.5617
51.jpg	litter	105.8568	339.2004	118.491	354.4711
51.jpg	litter	151.9098	220.7205	166.1888	237.283
51.jpg	litter	137.2546	470.8089	144.0037	479.9098
51.jpg	litter	202.133	571.6184	220.6546	584.3026
51.jpg	litter	275.896	607.6077	282.7939	616.466
51.jpg	litter	213.858	486.0476	220.675	497.9795
51.jpg	litter	168.2981	365.2634	176.8561	374.6134
51.jpg	litter	205.8718	141.3531	215.8004	147.3539
51.jpg	litter	145.9903	205.2346	156.8168	214.1149
51.jpg	litter	218.4532	51.10924	232.109	66.36207
51.jpg	litter	209.9006	250.1374	222.0557	260.8855
51.jpg	litter	184.6525	514.0924	193.1816	520.4623

52.jpg	litter	243.1807	645.092	257.073	662.189
52.jpg	litter	369.2005	357.5368	381.1109	371.4334
52.jpg	litter	291.8389	531.0084	308.4512	548.3686
52.jpg	litter	353.1539	565.3722	367.1632	576.8349
52.jpg	litter	350.82	452.1164	360.3579	460.9109
52.jpg	litter	309.1383	678.704	318.7037	684.4233
52.jpg	litter	359.5409	736.4522	366.7635	741.6348
52.jpg	litter	364.0515	697.136	371.7231	704.1453
53.jpg	litter	191.473	483.2025	205.1111	495.3802
53.jpg	litter	245.9072	681.0212	255.0805	689.3279
53.jpg	litter	191.9452	440.9996	202.0133	449.6245
53.jpg	litter	225.4242	517.6813	235.5263	530.6852
53.jpg	litter	157.9923	132.5014	169.8892	141.1287
53.jpg	litter	122.9296	48.56324	133.9636	57.08969
54.jpg	litter	160.0824	481.4953	173.3134	504.6683
54.jpg	litter	211.5047	730.5231	224.1605	741.0264
54.jpg	litter	258.7478	588.3807	267.2877	599.0656
55.jpg	litter	186.8875	416.7813	196.8917	425.9196
55.jpg	litter	261.7818	316.8841	270.5847	323.5668
55.jpg	litter	205.5764	268.6472	213.7331	276.1274
55.jpg	litter	163.568	168.7346	178.5069	186.6194
55.jpg	litter	229.1012	368.9824	238.5631	382.4881
55.jpg	litter	187.6284	726.5875	195.0574	736.3092
55.jpg	litter	191.7262	481.8564	216.2174	506.3954
55.jpg	litter	165.5581	248.6917	173.0683	257.7379
55.jpg	litter	180.1438	504.4355	189.794	517.0779
56.jpg	litter	401.8532	199.614	412.7141	213.5456
56.jpg	litter	389.8699	726.0823	407.0753	744.2117
56.jpg	litter	338.9458	324.8561	346.4357	332.0586
56.jpg	litter	340.9646	468.8481	350.1176	480.3031
57.jpg	litter	171.6025	486.0381	264.5294	633.3477

57.jpg	litter	481.8763	194.8029	505.119	227.5079
57.jpg	litter	501.4866	609.5606	515.5969	620.9976
57.jpg	litter	332.7618	434.3905	351.2209	451.268
57.jpg	litter	179.9504	596.0086	212.7378	625.9396
58.jpg	litter	278.4691	442.2824	298.2589	467.4387
58.jpg	litter	475.6972	541.0536	496.3347	553.6308
58.jpg	litter	377.994	469.7188	398.301	480.3654
59.jpg	litter	398.1163	61.02496	412.1357	81.3907
6.jpg	litter	206.5565	702.1788	225.9387	727.386
63.jpg	litter	296.6282	336.4561	313.6878	348.2769
63.jpg	litter	435.1972	418.4155	481.3423	458.9667
63.jpg	litter	473.1274	494.1516	501.7397	538.8614
7.jpg	litter	299.1107	441.8231	350.1346	502.3475
7.jpg	litter	310.4757	441.8906	355.1053	491.0586
7.jpg	litter	373.0929	607.7117	410.6866	672.3075
7.jpg	litter	312.382	435.6972	351.6408	472.2338
7.jpg	litter	415.6615	741.0019	454.8722	786.1481
7.jpg	litter	212.8561	354.3408	234.8441	369.6189
7.jpg	litter	319.12	450.603	356.5881	486.9039
7.jpg	litter	311.6468	437.265	345.3718	463.3879
7.jpg	litter	247.0655	289.3812	268.1163	331.058
71.jpg	litter	220.4535	323.24	255.0289	348.8525
72.jpg	litter	412.5776	496.4383	424.8781	510.6345
72.jpg	litter	319.8701	182.9046	334.0308	199.1158
72.jpg	litter	279.031	336.5013	293.88	361.0261
72.jpg	litter	330.2157	405.8839	349.5034	422.9707
72.jpg	litter	279.6375	72.40716	297.78	91.30536
72.jpg	litter	339.2446	450.6672	354.0678	469.5865
72.jpg	litter	300.8056	583.8218	326.1139	599.3048
72.jpg	litter	320.3713	357.5612	349.4653	381.569
72.jpg	litter	277.8631	592.7001	300.0253	611.6174

72.jpg	litter	489.3395	294.5155	509.5337	315.6229
72.jpg	litter	232.2216	106.1182	255.9148	130.6448
72.jpg	litter	275.0154	266.3442	286.2541	281.9288
72.jpg	litter	354.8128	675.7055	365.5665	691.8395
72.jpg	litter	307.4933	516.339	326.4036	530.4673
72.jpg	litter	125.7083	422.9019	142.9918	443.713
72.jpg	litter	281.6205	308.5114	297.1768	321.3094
72.jpg	litter	323.3724	724.7383	338.1641	739.2005
72.jpg	litter	280.8927	467.26	299.3595	493.7831
72.jpg	litter	280.3084	38.94716	294.3341	50.115
72.jpg	litter	378.159	628.3299	384.8405	635.3821
72.jpg	litter	302.0502	115.3133	321.261	128.8152
72.jpg	litter	318.8485	317.4228	338.538	338.5585
72.jpg	litter	334.3929	702.4672	343.2255	714.4094
72.jpg	litter	290.9339	627.5747	306.5765	650.1585
73.jpg	litter	482.5215	57.30422	496.767	77.07222
73.jpg	litter	483.2645	442.48	497.9304	459.3816
73.jpg	litter	225.451	437.5206	244.4524	456.3309
73.jpg	litter	392.4329	450.3712	405.5432	460.4598
73.jpg	litter	380.5358	255.5377	392.9937	265.8711
73.jpg	litter	178.6331	751.5227	189.2505	761.3858
73.jpg	litter	519.2172	735.9502	534.7882	750.4354
73.jpg	litter	290.1994	471.8762	302.3603	486.3085
73.jpg	litter	257.1261	541.326	294.5281	589.936
73.jpg	litter	516.1823	501.5605	527.476	511.9659
73.jpg	litter	214.1419	481.1841	229.3282	499.0253
73.jpg	litter	567.5335	283.6566	584.3529	346.3355
73.jpg	litter	196.3732	678.9109	209.8055	691.3703
73.jpg	litter	380.7075	682.0804	395.0153	702.7834
73.jpg	litter	403.4605	377.413	411.7658	387.6384
73.jpg	litter	523.7307	708.0353	538.9579	722.9116

73.jpg	litter	348.7832	695.4174	369.1121	710.683
73.jpg	litter	368.8603	502.6663	382.6316	516.3825
73.jpg	litter	231.4688	566.4434	247.9225	574.3504
73.jpg	litter	399.3099	615.7959	414.3894	632.2791
73.jpg	litter	24.79479	394.3743	47.55654	403.4491
73.jpg	litter	174.2297	543.8858	191.218	560.0374
73.jpg	litter	308.7826	62.25559	325.6462	126.0333
74.jpg	litter	227.4556	472.1633	248.6174	499.2297
74.jpg	litter	290.5833	201.8289	302.335	217.7855
74.jpg	litter	344.8645	668.4309	361.2662	689.7431
74.jpg	litter	314.504	516.5536	332.0656	545.1287
74.jpg	litter	312.7121	567.7039	329.929	589.0119
74.jpg	litter	338.3609	496.3746	355.5283	512.8198
74.jpg	litter	370.3611	432.1039	388.3902	445.0026
74.jpg	litter	184.2677	14.37021	193.0446	31.19419
74.jpg	litter	268.9209	754.5015	287.4149	773.6642
74.jpg	litter	304.5411	698.1169	326.5348	719.2863
74.jpg	litter	305.715	137.6624	327.7905	153.3182
74.jpg	litter	319.9	418.6052	342.9262	435.0407
74.jpg	litter	319.3008	455.4738	331.1938	467.421
74.jpg	litter	305.8213	488.5536	319.2472	500.2211
74.jpg	litter	349.1679	252.0656	371.2091	280.2573
74.jpg	litter	169.1965	275.8104	180.0675	285.1786
74.jpg	litter	337.9057	398.1907	350.698	410.694
74.jpg	litter	354.8666	636.649	383.842	674.6611
74.jpg	litter	330.1768	161.1821	341.9152	173.5214
74.jpg	litter	281.4572	289.5034	293.1247	305.8777
74.jpg	litter	358.801	750.1031	391.3389	771.7151
75.jpg	litter	358.0603	262.0336	372.4192	273.6878
75.jpg	litter	187.7272	114.4151	195.9019	122.8667
75.jpg	litter	267.8024	236.9913	279.0207	246.9354

75.jpg	litter	339.6683	44.47286	354.2431	59.44346
75.jpg	litter	321.8486	335.9377	329.9892	342.1591
75.jpg	litter	350.1575	584.9579	359.0899	592.8691
75.jpg	litter	377.1614	513.457	388.6616	531.7959
75.jpg	litter	354.3009	148.4753	369.3539	161.684
75.jpg	litter	309.5895	286.6062	315.3607	291.7596
75.jpg	litter	312.7314	235.9478	319.6667	244.2052
75.jpg	litter	73.33608	384.5617	82.33384	391.808
75.jpg	litter	321.704	710.5761	334.5671	722.7219
75.jpg	litter	284.768	80.36916	295.3857	89.69262
75.jpg	litter	342.3247	446.0489	348.7165	455.7614
76.jpg	litter	293.2357	525.2396	302.9977	533.0355
78.jpg	litter	416.47	258.2885	425.7848	264.8393
79.jpg	litter	313.8355	351.8764	322.7363	360.2088
8.jpg	litter	411.5724	158.2952	431.6118	173.1448
80.jpg	litter	495.5608	21.85258	548.1905	115.4813
80.jpg	litter	361.1825	625.0246	378.2293	655.614
80.jpg	litter	479.7069	705.7154	493.2498	723.8714
80.jpg	litter	409.8124	704.7574	423.428	722.121
80.jpg	litter	301.4442	477.4499	436.8612	559.7398
80.jpg	litter	398.4493	368.4859	416.3304	385.4298
80.jpg	litter	449.7547	532.5905	493.1619	577.4408
80.jpg	litter	258.368	480.8486	272.4825	492.2345
80.jpg	litter	304.3066	546.7207	314.3906	556.2039
80.jpg	litter	490.0391	760.1191	511.9229	779.8193
80.jpg	litter	312.923	471.4446	350.9099	499.5252
80.jpg	litter	231.0531	313.9434	237.7665	321.8079
80.jpg	litter	254.3939	79.39978	540.652	708.5409
80.jpg	litter	438.6024	623.2562	484.9358	652.3522
80.jpg	litter	447.752	577.0541	461.6925	599.8548
80.jpg	litter	201.5357	679.3845	212.6336	690.1725

80.jpg	litter	67.41613	208.2763	78.64425	223.8696
80.jpg	litter	444.2793	499.4649	455.7714	507.2967
80.jpg	litter	426.0008	324.6774	438.3959	342.5832
80.jpg	litter	291.2615	468.8254	304.3045	483.2658
80.jpg	litter	372.8534	416.1133	384.6604	430.1231
80.jpg	litter	227.7321	549.6812	236.5851	560.3665
80.jpg	litter	112.1903	165.8284	119.9461	173.7214
80.jpg	litter	224.1348	496.095	235.2875	510.575
80.jpg	litter	329.1377	306.103	339.1822	315.1702
80.jpg	litter	227.8477	647.2046	235.7264	654.9204
81.jpg	litter	299.4325	485.1828	342.839	544.4397
81.jpg	litter	390.793	58.95012	418.2857	106.7297
81.jpg	litter	273.6052	340.082	287.3428	349.7739
81.jpg	litter	200.3488	506.1691	208.0935	516.6199
81.jpg	litter	366.0486	536.06	378.5314	543.425
81.jpg	litter	238.481	467.407	248.2641	481.846
81.jpg	litter	399.287	209.0525	410.4604	219.3333
81.jpg	litter	580.8629	7.082132	592.8268	14.27611
81.jpg	litter	384.5604	111.4544	393.2077	121.7572
81.jpg	litter	354.3358	262.9676	372.6055	275.213
81.jpg	litter	348.7252	370.0329	359.6801	378.4371
81.jpg	litter	330.1039	348.8258	340.6899	358.8235
81.jpg	litter	373.4548	334.0327	383.7735	344.5083
81.jpg	litter	272.168	706.327	282.3568	721.7661
81.jpg	litter	156.194	767.3205	191.322	799.7716
81.jpg	litter	372.6697	191.2385	387.0917	204.3058
81.jpg	litter	319.5946	771.3216	329.588	778.6834
82.jpg	litter	240.5438	570.5126	246.7576	577.2771
82.jpg	litter	240.2912	572.1543	246.4973	578.8647
83.jpg	litter	362.5753	400.4186	462.1057	508.9322
83.jpg	litter	74.50449	646.4192	85.10321	655.9407

83.jpg	litter	55.83903	560.2862	67.68151	574.0364
83.jpg	litter	352.7876	489.7386	363.511	506.2013
83.jpg	litter	398.5619	402.3555	417.8057	415.9569
83.jpg	litter	360.3166	525.9302	373.5199	538.2301
83.jpg	litter	209.2507	562.4887	218.7541	570.4412
84.jpg	litter	234.9026	287.3567	350.327	416.3912
84.jpg	litter	204.0408	394.5946	222.9144	413.4423
84.jpg	litter	120.4333	517.7489	134.2998	526.7201
84.jpg	litter	212.844	458.6793	232.3827	476.1258
84.jpg	litter	285.3613	293.4313	306.846	312.5185
84.jpg	litter	451.9141	269.7421	463.8059	282.8249
84.jpg	litter	154.989	472.6173	170.9822	486.2569
84.jpg	litter	236.6874	406.7051	258.6836	427.7577
85.jpg	litter	297.2666	309.8935	314.4168	343.134
85.jpg	litter	234.2965	304.2087	243.501	315.6046
85.jpg	litter	377.0522	319.6466	398.0731	342.0801
85.jpg	litter	302.695	414.1444	359.8478	466.7375
85.jpg	litter	416.8098	332.9466	433.8254	356.5505
85.jpg	litter	385.7596	488.1245	396.7384	500.5634
85.jpg	litter	286.8443	576.7349	297.1707	586.1555
85.jpg	litter	480.2928	312.4206	517.6257	417.141
85.jpg	litter	357.5561	450.261	380.5134	466.5196
85.jpg	litter	257.2611	496.9713	267.181	512.8216
85.jpg	litter	304.8509	422.1622	318.7787	434.5263
85.jpg	litter	420.102	503.9447	431.7115	514.1279
85.jpg	litter	391.241	99.18134	402.5429	112.1787
85.jpg	litter	482.6257	506.919	492.8903	518.8959
85.jpg	litter	408.0965	372.6005	417.5693	389.7422
85.jpg	litter	307.9137	375.8439	319.7901	385.1634
86.jpg	litter	223.3934	334.8829	239.6717	346.9575
86.jpg	litter	311.9953	420.8172	328.1815	463.1107

86.jpg	litter	241.0317	179.9756	250.5119	192.1917
86.jpg	litter	351.047	708.4767	359.2687	720.6059
86.jpg	litter	165.52	312.7162	176.4251	332.7204
86.jpg	litter	323.8455	497.1314	335.5717	507.0171
86.jpg	litter	308.0229	371.3177	328.7379	410.2529
86.jpg	litter	387.8404	396.0406	397.3386	406.7012
86.jpg	litter	321.7838	565.6106	332.2816	578.1736
86.jpg	litter	211.8819	421.3283	224.1187	435.4454
86.jpg	litter	141.6861	549.4894	149.7812	558.7318
86.jpg	litter	360.6038	421.6633	373.3876	432.8131
86.jpg	litter	155.2453	447.1632	165.9421	463.4761
88.jpg	litter	172.5836	225.5028	196.0432	244.2952
9.jpg	litter	449.7043	238.135	467.888	254.6661
93.jpg	litter	201.715	721.0092	210.0458	728.7576
94.jpg	litter	407.4712	180.267	428.3457	218.8362
94.jpg	litter	411.1815	240.8846	420.7106	249.3499
97.jpg	litter	235.4903	385.2113	247.45	398.6388
97.jpg	litter	218.2329	469.0538	226.8917	481.4768
97.jpg	litter	227.4402	226.8842	236.2898	235.6874
97.jpg	litter	76.70984	176.3865	85.4162	184.0648

Table 7-3 The result of the prediction from model 8 (Trained with Faster R-CNN Resnet 101)

image_name	class	ymin	xmin	ymax	xmax
11.jpg	litter	285.234	114.4989	296.2128	135.7301
11.jpg	litter	283.3205	469.6256	290.8243	480.5289
11.jpg	litter	191.5323	203.8793	198.7633	209.8719
18.jpg	litter	145.8266	235.7139	162.08	270.8629
18.jpg	litter	176.6779	296.6507	193.6273	305.584
19.jpg	litter	39.28951	381.6224	52.74455	398.9213
21.jpg	litter	280.9911	375.3157	299.9815	390.9079

21.jpg	litter	258.1445	361.0882	267.3769	371.612
23.jpg	litter	455.0637	144.8361	478.439	179.2256
23.jpg	litter	211.8072	570.4767	227.2257	594.4264
26.jpg	litter	235.0568	509.6968	242.7877	519.3847
27.jpg	litter	246.4179	223.4804	267.3161	247.7014
28.jpg	litter	383.9563	463.8813	393.8004	473.1997
29.jpg	litter	372.2571	453.2803	386.2904	469.1715
30.jpg	litter	173.8021	427.7364	184.9327	438.4227
32.jpg	litter	198.0867	294.0471	216.9488	312.4564
32.jpg	litter	30.72217	642.223	39.11502	647.994
33.jpg	litter	234.1123	383.9668	244.0303	394.2218
33.jpg	litter	305.6738	327.7255	316.4686	338.0784
34.jpg	litter	264.5246	414.6119	281.058	435.3714
35.jpg	litter	493.3999	47.50138	505.9988	63.49564
35.jpg	litter	510.7029	545.2976	527.6382	561.991
35.jpg	litter	494.2333	125.2648	516.5045	140.3592
35.jpg	litter	273.369	301.4266	299.294	331.4618
35.jpg	litter	262.2017	329.1981	306.9611	503.4321
36.jpg	litter	347.859	366.7805	362.7052	381.4185
36.jpg	litter	341.1655	257.0161	352.3738	270.9718
36.jpg	litter	167.9869	659.0251	176.477	666.7993
37.jpg	litter	311.2222	407.6947	328.4271	421.0874
37.jpg	litter	285.2287	344.7847	297.3676	354.5662
37.jpg	litter	343.7858	664.2261	352.5182	672.709
38.jpg	litter	317.6184	530.6979	327.6571	541.4374
39.jpg	litter	299.6493	211.8153	312.1457	223.1386
39.jpg	litter	325.6534	329.6294	336.4221	339.566
39.jpg	litter	279.6216	310.0009	292.2359	321.5112
39.jpg	litter	273.3647	209.5037	290.4844	226.1548
39.jpg	litter	316.9106	357.8614	327.3121	368.9084
4.jpg	litter	252.5736	227.7958	271.4541	257.761

40.jpg	litter	350.2263	289.0334	366.6212	308.9019
40.jpg	litter	389.6007	348.5754	402.0834	363.3633
41.jpg	litter	347.8486	500.851	362.3917	515.5827
41.jpg	litter	301.1735	700.196	312.6471	710.1132
41.jpg	litter	259.0561	383.5304	272.7913	400.8682
41.jpg	litter	394.6009	351.6243	411.6294	371.5771
41.jpg	litter	256.0323	515.625	267.4288	526.7723
41.jpg	litter	357.1763	110.7641	388.713	150.9792
42.jpg	litter	286.981	265.277	297.6235	279.0829
42.jpg	litter	273.2898	367.4654	279.3818	377.8022
44.jpg	litter	127.219	0	220.2307	383.4808
44.jpg	litter	126.213	432.6646	193.1642	634.3226
45.jpg	litter	147.3371	169.045	158.783	180.2596
45.jpg	litter	276.171	226.8162	286.3919	238.119
45.jpg	litter	270.9265	245.3353	278.9401	255.7961
46.jpg	litter	151.707	243.9346	189.1179	268.4442
46.jpg	litter	249.3988	459.7053	262.1186	469.774
46.jpg	litter	187.5571	353.9229	200.7286	362.507
46.jpg	litter	288.2933	377.431	315.9312	410.4383
46.jpg	litter	131.0406	368.3547	163.9337	423.585
46.jpg	litter	189.3827	789.1214	201.5058	798.1886
47.jpg	litter	208.7779	322.562	224.4854	342.2011
47.jpg	litter	98.31288	166.344	109.5086	173.9201
48.jpg	litter	198.2698	402.5136	213.5757	417.5179
48.jpg	litter	184.8034	421.9955	194.6365	430.7406
48.jpg	litter	198.6071	506.2011	208.725	519.9583
48.jpg	litter	213.5053	625.9138	252.6953	662.189
48.jpg	litter	215.1267	277.8854	224.9377	287.3657
48.jpg	litter	189.489	457.3326	200.7478	464.8147
48.jpg	litter	215.0959	511.486	228.2411	521.2983
48.jpg	litter	236.686	735.9369	253.2872	745.8332

49.jpg	litter	255.3582	610.5053	266.8798	624.4891
49.jpg	litter	291.542	301.2078	308.1399	314.837
49.jpg	litter	346.4061	30.84574	364.4011	46.9383
49.jpg	litter	288.7194	404.4938	302.3888	415.091
49.jpg	litter	311.4281	374.9344	320.7312	383.4524
50.jpg	litter	263.0671	418.131	277.1913	435.3148
50.jpg	litter	259.6391	604.5346	274.0052	618.6572
50.jpg	litter	259.3323	382.0456	271.7006	395.9794
50.jpg	litter	351.3789	464.4977	361.5622	483.477
51.jpg	litter	152.5376	223.5205	167.7722	239.2495
51.jpg	litter	206.0792	569.4357	221.5692	581.9732
51.jpg	litter	218.9685	50.65491	232.1624	63.04327
51.jpg	litter	210.6914	248.956	222.6181	261.1396
51.jpg	litter	137.7684	470.0497	147.1367	479.7652
51.jpg	litter	107.0642	339.9914	118.4116	352.6803
51.jpg	litter	306.2207	319.1767	319.184	330.5745
51.jpg	litter	215.7754	489.8151	222.9177	497.6516
51.jpg	litter	224.8879	376.8718	231.7398	387.8522
51.jpg	litter	143.3659	399.1875	152.7437	414.0458
51.jpg	litter	168.6476	366.3163	178.0964	374.5262
51.jpg	litter	285.0229	324.0422	296.4731	334.7851
51.jpg	litter	235.2741	142.2979	241.9628	150.308
52.jpg	litter	367.6231	355.9774	377.2786	369.6972
52.jpg	litter	294.1102	530.3451	308.8346	549.3907
52.jpg	litter	429.8652	647.3827	441.9066	657.4438
52.jpg	litter	352.4287	454.7805	360.0927	461.9986
52.jpg	litter	365.6358	696.5947	374.0983	704.907
52.jpg	litter	244.8282	643.2055	257.0308	659.6356
52.jpg	litter	354.3357	564.1369	365.7034	577.5382
52.jpg	litter	450.7869	643.8902	463.1939	655.4556
52.jpg	litter	374.2964	631.2562	386.9921	659.2731

52.jpg	litter	273.5784	778.2618	281.1638	787.8153
52.jpg	litter	281.7201	710.0004	287.7052	722.3756
52.jpg	litter	351.3833	607.257	357.8428	614.7148
53.jpg	litter	446.588	402.0297	460.6336	423.7569
53.jpg	litter	156.393	131.3066	171.0964	143.1083
53.jpg	litter	192.7373	484.5002	204.9315	494.181
53.jpg	litter	227.822	518.7738	236.3913	529.6595
53.jpg	litter	244.8306	683.3576	253.1615	691.554
53.jpg	litter	162.3466	341.2258	171.836	355.5212
53.jpg	litter	197.0801	439.8635	205.6466	448.0394
53.jpg	litter	149.4949	20.78919	158.1966	29.44857
54.jpg	litter	160.4806	480.1125	172.9014	504.3495
54.jpg	litter	211.5804	731.9523	225.0224	742.0054
54.jpg	litter	286.2808	549.7633	294.8202	559.3737
55.jpg	litter	229.8872	370.7541	239.6166	382.118
55.jpg	litter	222.2697	114.5204	231.5519	124.3335
55.jpg	litter	188.0612	414.0739	196.7907	423.7273
55.jpg	litter	167.9774	174.3856	179.1866	192.7556
55.jpg	litter	206.9518	266.8471	214.2379	274.4066
55.jpg	litter	263.7357	315.7751	271.3463	324.0078
55.jpg	litter	227.3351	546.8995	236.3816	557.7012
55.jpg	litter	191.283	773.2472	200.9867	787.1276
55.jpg	litter	166.81	248.0263	175.9705	257.5846
55.jpg	litter	181.1009	505.2537	189.2919	515.5701
56.jpg	litter	400.7249	202.7142	410.8532	219.0199
56.jpg	litter	388.8223	729.0413	404.533	743.7541
56.jpg	litter	342.0107	471.5142	349.5841	481.7065
56.jpg	litter	318.9009	565.7996	329.2408	574.9361
56.jpg	litter	358.6112	197.9122	375.9341	230.3891
56.jpg	litter	343.4834	584.3348	354.2375	592.4287
57.jpg	litter	331.7882	433.5569	350.7712	452.5875

57.jpg	litter	482.4226	198.9594	506.2573	228.9635
57.jpg	litter	187.8856	471.2635	274.1486	623.4814
57.jpg	litter	171.6724	530.9597	199.5174	579.1923
57.jpg	litter	183.6692	599.1844	206.5089	620.8252
59.jpg	litter	397.2371	64.09008	409.1963	81.47008
6.jpg	litter	209.5633	701.2229	223.7945	731.1618
63.jpg	litter	296.8011	336.7524	311.7551	349.5435
63.jpg	litter	311.4539	185.2567	365.5967	241.7814
63.jpg	litter	477.8691	498.9129	514.5188	543.4413
63.jpg	litter	507.5143	754.6041	517.8767	763.6233
63.jpg	litter	397.3485	146.6745	406.5782	155.83
64.jpg	litter	257.3267	687.874	311.0139	774.5089
66.jpg	litter	44.34915	284.8104	54.94297	294.6945
7.jpg	litter	414.8466	737.9099	453.2134	784.2002
7.jpg	litter	302.2808	437.4296	346.4244	501.6616
7.jpg	litter	368.0265	604.5566	403.5195	674.9749
7.jpg	litter	314.1481	456.8173	352.588	490.1865
7.jpg	litter	251.4964	287.7249	260.253	296.2315
70.jpg	litter	507.1303	316.5632	530.6674	382.5583
71.jpg	litter	223.4874	320.0724	254.804	350.723
72.jpg	litter	320.2308	360.1329	348.2453	379.538
72.jpg	litter	280.8944	73.92903	296.0859	91.02021
72.jpg	litter	277.6732	336.8566	293.1547	357.9041
72.jpg	litter	233.1347	108.6554	252.4343	129.9094
72.jpg	litter	334.0808	407.6087	350.7793	423.9944
72.jpg	litter	306.8771	517.5436	326.2795	532.6027
72.jpg	litter	412.979	496.5568	424.655	513.3164
72.jpg	litter	283.2308	470.6873	297.4584	494.2974
72.jpg	litter	308.3937	581.9215	325.109	598.6672
72.jpg	litter	329.9408	312.4894	349.924	334.9622
72.jpg	litter	491.9799	292.1753	509.8375	313.8184

72.jpg	litter	321.8548	183.9964	333.2846	203.2213
72.jpg	litter	276.6404	594.8468	295.7602	607.3945
72.jpg	litter	281.0169	40.13348	295.5998	51.32
72.jpg	litter	289.6562	624.6829	307.4299	648.0192
72.jpg	litter	100.5106	252.4009	113.1186	268.8929
72.jpg	litter	317.0858	332.8562	330.9488	348.9108
72.jpg	litter	325.3109	723.0793	337.7944	740.1414
72.jpg	litter	353.3462	675.4282	367.5439	694.7362
72.jpg	litter	333.5484	702.8858	342.5161	715.4207
72.jpg	litter	286.8572	158.7353	298.5166	190.9629
72.jpg	litter	283.3345	313.826	297.4924	324.5792
72.jpg	litter	283.9472	587.6319	302.0834	602.4471
72.jpg	litter	312.8156	83.97122	329.0594	109.3453
73.jpg	litter	402.3085	376.6591	410.9824	387.631
73.jpg	litter	290.5883	473.1895	302.2499	489.6787
73.jpg	litter	569.2435	282.7966	583.5804	349.9211
73.jpg	litter	484.2773	59.69334	499.9149	78.50927
73.jpg	litter	179.9739	750.1306	189.0508	760.7792
73.jpg	litter	525.5654	705.6186	539.8502	716.828
73.jpg	litter	382.332	683.7048	396.0986	701.7422
73.jpg	litter	368.4448	499.8922	380.6895	514.502
73.jpg	litter	350.2042	695.7869	366.8632	717.1827
73.jpg	litter	258.9981	537.836	293.0731	593.3954
73.jpg	litter	484.8978	445.1324	495.3287	459.6252
73.jpg	litter	517.3942	502.9167	526.9065	513.2841
73.jpg	litter	23.47407	393.6709	41.37318	404.5794
73.jpg	litter	382.067	257.5958	394.5548	270.7377
73.jpg	litter	226.8792	439.9068	243.46	457.1754
73.jpg	litter	391.7524	453.4155	403.1372	463.1316
73.jpg	litter	410.7839	632.4407	424.4888	666.4727
73.jpg	litter	241.7182	718.3053	252.6614	728.0587

73.jpg	litter	235.4876	516.3779	246.9882	527.6161
73.jpg	litter	236.0071	515.2462	251.2427	530.5667
73.jpg	litter	472.04	164.9482	482.2086	176.645
73.jpg	litter	288.5824	364.636	300.0345	378.0236
73.jpg	litter	215.9732	484.3225	232.3787	497.2448
74.jpg	litter	356.705	633.5402	382.1214	673.7936
74.jpg	litter	230.962	474.1643	249.0253	497.7588
74.jpg	litter	289.8843	205.2851	302.4311	218.7835
74.jpg	litter	303.1154	139.8881	326.8048	157.302
74.jpg	litter	314.9661	564.5195	331.1561	584.5193
74.jpg	litter	281.9428	292.2021	293.4356	305.0641
74.jpg	litter	277.7748	142.5931	287.8919	153.5466
74.jpg	litter	331.1434	199.1167	342.8278	209.7586
74.jpg	litter	319.9432	419.7967	341.129	435.9853
74.jpg	litter	368.2502	430.8692	384.3267	446.9326
74.jpg	litter	171.824	278.0689	183.6093	286.3775
74.jpg	litter	268.4327	755.624	286.9062	772.1874
74.jpg	litter	350.6058	748.2644	385.6172	768.8119
74.jpg	litter	339.7728	399.1765	350.2508	408.4927
74.jpg	litter	304.8471	355.8702	314.144	366.9216
74.jpg	litter	308.5156	486.5397	317.9663	497.8047
74.jpg	litter	318.8184	365.1332	330.5655	373.6907
74.jpg	litter	315.1071	518.0808	330.7003	538.3065
74.jpg	litter	307.2099	699.6138	324.8988	720.9583
74.jpg	litter	319.7131	456.4145	330.6161	465.6719
75.jpg	litter	189.0096	112.5966	197.7534	123.0206
75.jpg	litter	338.9678	41.2928	353.9128	55.69878
75.jpg	litter	376.7444	515.2171	386.814	527.8613
75.jpg	litter	357.2304	262.4402	371.4334	274.3194
75.jpg	litter	323.0164	702.2964	338.2264	722.914
75.jpg	litter	74.85404	381.5854	83.47887	391.9402

75.jpg	litter	348.6666	581.2769	356.6769	589.5691
75.jpg	litter	268.2089	240.1947	278.4435	248.0234
75.jpg	litter	332.4517	642.4069	343.3588	667.3637
75.jpg	litter	341.3957	445.5199	348.2117	454.9792
75.jpg	litter	318.5414	337.9758	329.5256	346.8256
76.jpg	litter	291.9676	522.8092	303.0707	530.0664
77.jpg	litter	381.7265	585.3062	389.7905	596.816
78.jpg	litter	412.9599	259.328	421.8219	267.199
79.jpg	litter	206.3183	511.9583	215.5286	521.9668
8.jpg	litter	410.2795	163.0247	426.8252	176.6488
80.jpg	litter	489.4686	32.04902	546.1748	106.6576
80.jpg	litter	410.8068	708.2973	422.4163	721.7431
80.jpg	litter	68.54602	212.8586	78.41214	227.8463
80.jpg	litter	446.1492	577.0111	459.8932	595.7587
80.jpg	litter	361.3779	223.8212	370.3801	235.5119
80.jpg	litter	478.9206	707.1497	494.4006	723.3403
80.jpg	litter	406.0269	24.65774	419.5996	39.15742
80.jpg	litter	326.8073	479.5886	420.9232	561.4338
80.jpg	litter	328.9617	307.9465	338.6105	315.7305
80.jpg	litter	425.7185	324.8673	440.115	338.9033
80.jpg	litter	354.5257	95.59847	363.8107	107.8604
80.jpg	litter	364.5779	630.127	378.3092	650.7076
80.jpg	litter	131.2018	710.3494	139.4546	719.2816
80.jpg	litter	399.8015	368.5217	416.7576	386.6915
80.jpg	litter	471.3541	544.7068	492.5016	567.0065
80.jpg	litter	383.5838	497.0444	419.3558	538.829
80.jpg	litter	430.4514	447.4048	443.311	461.9836
80.jpg	litter	446.2541	290.2318	461.0281	303.09
80.jpg	litter	463.1242	167.4789	478.4485	200.9882
80.jpg	litter	440.0298	209.6816	486.2753	279.4748
81.jpg	litter	386.122	57.48222	416.4666	105.5189

81.jpg	litter	298.6082	488.2411	343.3218	541.4527
81.jpg	litter	299.2579	392.5165	329.3949	421.9232
81.jpg	litter	371.4256	333.3815	381.7351	343.0199
81.jpg	litter	399.1017	208.6697	410.869	219.9519
81.jpg	litter	353.6563	260.1439	372.6161	276.657
81.jpg	litter	309.1251	323.2384	320.2695	335.1084
81.jpg	litter	277.9674	336.4131	290.9504	350.6237
81.jpg	litter	366.0619	532.8284	378.2795	541.1501
81.jpg	litter	342.9429	211.4428	350.4848	220.523
81.jpg	litter	582.1686	6.16924	593.4028	19.18666
81.jpg	litter	200.9674	508.0599	208.7828	515.6015
81.jpg	litter	368.013	628.6601	378.7045	638.653
81.jpg	litter	327.6606	434.3778	339.0486	447.27
81.jpg	litter	335.9089	291.323	351.886	306.722
83.jpg	litter	56.41807	557.6864	68.02462	572.9445
83.jpg	litter	362.9175	394.6538	450.1233	516.6062
83.jpg	litter	74.51081	645.3002	84.0116	654.5764
83.jpg	litter	351.9106	490.9335	366.3796	506.7019
83.jpg	litter	365.6873	522.6126	374.917	534.7456
83.jpg	litter	567.7676	102.6142	584.0092	118.7661
84.jpg	litter	219.7915	281.9455	361.8418	452.9859
84.jpg	litter	206.9976	396.5085	223.187	414.6254
84.jpg	litter	286.2105	291.1458	304.6368	312.0084
84.jpg	litter	461.5437	263.4344	544.3503	337.5684
84.jpg	litter	242.8662	400.2737	264.8463	420.7703
84.jpg	litter	219.6887	463.5946	233.327	472.9548
85.jpg	litter	483.1534	505.7899	494.3757	518.6556
85.jpg	litter	391.8008	98.37896	401.5434	109.8384
85.jpg	litter	297.6919	307.4094	315.8495	345.6699
85.jpg	litter	445.6895	321.26	460.8366	334.4927
85.jpg	litter	515.6352	604.0894	528.0194	617.6749

85.jpg	litter	234.2231	301.026	244.625	313.8751
85.jpg	litter	516.4196	510.5888	528.1425	524.0592
85.jpg	litter	420.0662	502.8948	432.14	514.8513
85.jpg	litter	442.6217	523.3671	452.7396	534.5479
85.jpg	litter	376.4066	322.5181	398.3579	340.5432
85.jpg	litter	286.0463	571.908	296.2303	581.4398
85.jpg	litter	469.9446	304.8373	527.5096	490.6916
85.jpg	litter	364.3024	450.0729	382.4456	468.6363
85.jpg	litter	276.0617	643.4024	284.1681	652.7365
85.jpg	litter	303.5978	422.2415	356.6491	467.9145
85.jpg	litter	302.5648	319.146	388.5602	475.0111
86.jpg	litter	241.3259	185.0647	248.9438	195.1877
86.jpg	litter	327.7798	539.5236	338.4429	551.442
86.jpg	litter	379.2955	544.8992	389.9391	563.1319
86.jpg	litter	297.9423	484.5578	310.016	495.2014
86.jpg	litter	387.4074	393.4128	398.323	403.8076
86.jpg	litter	225.8507	334.0532	240.3257	348.0555
86.jpg	litter	168.4752	312.1339	179.2238	331.0824
86.jpg	litter	324.5558	495.4673	334.2625	504.1046
86.jpg	litter	269.8431	489.3256	280.9	501.2774
86.jpg	litter	170.536	352.3736	223.1749	436.4116
86.jpg	litter	163.7526	323.0295	228.7593	435.269
86.jpg	litter	151.6256	499.1128	162.0926	507.8383
86.jpg	litter	362.447	418.6971	372.8051	431.4576
86.jpg	litter	278.19	338.1498	296.5763	363.3795
86.jpg	litter	322.0324	561.9921	330.7688	574.0588
86.jpg	litter	354.1579	711.425	360.7453	718.8688
86.jpg	litter	128.3715	380.8957	138.5979	392.9588
86.jpg	litter	158.6933	440.9603	168.0079	465.7952
86.jpg	litter	254.5584	470.2135	265.1877	482.0116
86.jpg	litter	260.3238	523.0625	270.4832	530.9471

88.jpg	litter	175.955	227.8435	195.3455	247.5778
9.jpg	litter	451.1066	240.2679	466.9428	255.0786
94.jpg	litter	411.4422	237.2877	418.7582	247.4138
97.jpg	litter	236.5227	385.7114	248.1417	399.5824
97.jpg	litter	217.6614	470.9426	226.4644	480.6622
97.jpg	litter	228.3913	230.7209	236.2394	239.4029
97.jpg	litter	79.10261	181.7799	88.46748	189.1447

Table 7-4 The result of the prediction from model 8 (Trained with Faster R-CNN Inception Resnet v2)

image_name	class	ymin	xmin	ymax	xmax
11.jpg	litter	271.3057	194.9902	281.5258	204.3312
19.jpg	litter	42.08268	384.0085	51.61999	397.5613
19.jpg	litter	162.3884	402.8603	176.5132	419.041
21.jpg	litter	258.1964	361.0255	267.4117	372.3869
21.jpg	litter	283.8701	373.7133	301.2978	389.9441
23.jpg	litter	231.2699	186.005	241.1717	198.2516
26.jpg	litter	235.3118	508.4254	243.7518	520.6668
27.jpg	litter	246.0347	225	266.7147	248.9584
28.jpg	litter	383.1347	462.0317	392.7894	471.9285
29.jpg	litter	374.7473	453.7627	387.4628	466.4677
30.jpg	litter	175.0979	426.2619	185.3459	437.4183
32.jpg	litter	31.7775	642.0054	40.07878	649.7032
32.jpg	litter	196.6146	379.7858	205.2924	388.6869
32.jpg	litter	200.2408	298.5135	217.3872	315.7948
33.jpg	litter	233.8035	383.3468	245.3773	395.5981
34.jpg	litter	264.4106	417.107	283.7502	435.0286
35.jpg	litter	495.7801	48.21066	509.5418	62.49005
35.jpg	litter	507.4576	546.1346	528.5924	564.1639
36.jpg	litter	168.3189	658.0724	177.3041	666.3757
36.jpg	litter	341.1962	259.5175	350.0935	271.7052

36.jpg	litter	344.3178	366.2564	364.4747	396.8122
37.jpg	litter	285.005	344.0563	297.7128	357.4052
37.jpg	litter	313.3274	408.3211	327.1117	422.5863
38.jpg	litter	318.0816	529.177	327.9576	537.7159
39.jpg	litter	276.7444	212.8592	290.2455	225.1623
39.jpg	litter	279.7832	310.5168	293.427	323.0705
39.jpg	litter	300.4163	214.0701	312.0634	227.3727
39.jpg	litter	317.7636	357.8413	328.4077	368.5364
39.jpg	litter	325.2233	334.6065	334.6241	342.9704
4.jpg	litter	252.562	229.0854	272.253	257.4569
40.jpg	litter	349.2913	293.2696	363.7583	309.4658
40.jpg	litter	388.902	349.5342	404.025	364.942
41.jpg	litter	255.7908	515.506	268.6801	524.5305
41.jpg	litter	258.2535	384.1383	272.1845	399.6616
41.jpg	litter	301.053	699.7873	313.4115	709.069
41.jpg	litter	330.076	360.4526	341.546	375.7287
41.jpg	litter	357.8698	111.0672	388.3224	151.5856
42.jpg	litter	272.2073	369.6397	278.8278	377.9142
42.jpg	litter	286.2414	267.7473	296.9392	280.4092
43.jpg	litter	358.1018	411.8011	374.976	428.4601
45.jpg	litter	147.6806	169.1905	159.0315	179.4009
45.jpg	litter	276.1286	229.0835	285.4783	236.7453
45.jpg	litter	416.5272	632.7448	450.3184	666.9611
46.jpg	litter	128.4706	368.9969	167.7284	424.6574
46.jpg	litter	151.0272	242.8912	189.1601	268.0063
46.jpg	litter	187.8949	352.6343	202.6914	365.3783
46.jpg	litter	251.3657	459.2232	262.6357	469.2924
46.jpg	litter	280.561	374.4188	310.429	409.6448
47.jpg	litter	99.57688	164.7222	109.1831	173.8279
47.jpg	litter	208.333	324.3515	222.867	343.3707
48.jpg	litter	183.7524	423.8553	196.6998	437.8829

48.jpg	litter	189.5384	451.2649	201.4672	462.9965
48.jpg	litter	199.0385	502.0953	209.8262	520.7114
48.jpg	litter	199.3876	572.0029	220.7056	597.8127
48.jpg	litter	209.0027	623.2461	249.8537	662.7138
48.jpg	litter	214.162	511.6154	225.8796	522.6529
48.jpg	litter	217.1154	280.9283	227.2031	289.7688
48.jpg	litter	219.429	220.415	237.4163	241.0978
48.jpg	litter	234.2824	734.1017	255.9945	745.7674
49.jpg	litter	254.5759	610.3289	267.5435	624.487
49.jpg	litter	288.6969	403.2947	303.7168	414.6436
49.jpg	litter	291.0644	302.6906	306.8599	314.1316
49.jpg	litter	311.7814	373.7969	325.1631	393.1073
49.jpg	litter	344.8664	33.32657	362.2196	48.71208
50.jpg	litter	215.9933	644.1054	226.5984	653.6236
50.jpg	litter	258.724	604.1031	275.2282	620.9621
50.jpg	litter	263.4279	414.1009	278.3938	436.2312
51.jpg	litter	107.2387	338.5002	120.2148	355.1977
51.jpg	litter	139.8199	471.343	146.2264	479.6016
51.jpg	litter	150.6468	220.1688	169.5585	240.7699
51.jpg	litter	203.4658	567.7255	223.975	581.0565
51.jpg	litter	209.6131	643.8694	226.7241	659.714
51.jpg	litter	211.5354	251.0583	225.215	261.3228
51.jpg	litter	259.5868	682.8613	270.4908	696.8454
51.jpg	litter	305.1656	322.9791	317.4396	332.5302
52.jpg	litter	244.173	641.1451	257.7256	658.4616
52.jpg	litter	273.969	776.9397	279.8796	787.4331
52.jpg	litter	291.4424	523.0367	309.0887	545.3158
52.jpg	litter	350.6067	452.9054	360.6314	463.2101
52.jpg	litter	354.0305	562.5959	365.4028	573.1086
52.jpg	litter	367.7207	358.2945	377.5999	372.31
52.jpg	litter	451.307	643.9913	465.1478	654.422

53.jpg	litter	124.712	50.75148	134.9656	56.93123
53.jpg	litter	157.6936	133.4408	171.1095	143.5681
53.jpg	litter	168.5401	271.1479	177.501	278.7362
53.jpg	litter	192.3145	482.5833	207.1119	493.6647
53.jpg	litter	194.941	440.1312	203.7288	449.7766
53.jpg	litter	196.9464	564.9242	209.6372	573.2499
53.jpg	litter	216.5149	432.5202	225.9558	440.8417
53.jpg	litter	228.2128	518.2541	237.6426	529.1459
53.jpg	litter	245.7788	682.4949	253.9935	691.1543
53.jpg	litter	251.2596	469.9107	261.1649	479.1379
53.jpg	litter	444.5169	402.5324	457.5266	421.2959
54.jpg	litter	159.6358	477.5153	172.5016	503.9304
54.jpg	litter	173.2507	638.3843	182.1774	647.1503
54.jpg	litter	173.9012	638.1449	181.9004	643.5146
54.jpg	litter	212.8138	731.5701	225.8593	742.3733
54.jpg	litter	257.9724	587.4327	267.624	598.5812
54.jpg	litter	263.4754	486.3135	270.9238	495.2982
54.jpg	litter	285.8896	550.8852	296.2228	559.2254
55.jpg	litter	164.5994	249.9886	180.1564	262.4179
55.jpg	litter	184.5075	585.1085	195.2461	596.2322
55.jpg	litter	188.7721	415.1015	197.438	423.4214
55.jpg	litter	189.8555	725.6645	198.0719	736.0959
55.jpg	litter	263.3158	315.7375	271.2052	324.5235
56.jpg	litter	318.6074	564.5129	328.4706	576.5121
56.jpg	litter	320.0725	596.8033	332.0881	608.6161
56.jpg	litter	336.4152	324.7497	344.3955	333.2818
56.jpg	litter	338.9726	469.8967	349.7541	480.5231
56.jpg	litter	356.6061	191.6612	380.0735	224.5327
56.jpg	litter	388.6748	727.8399	405.0403	744.2874
57.jpg	litter	167.8806	477.8002	276.5786	628.0528
57.jpg	litter	333.8484	431.9124	350.2096	450.1411

57.jpg	litter	483.9397	202.205	507.0568	231.8514
58.jpg	litter	278.6857	438.4723	300.0839	469.8277
58.jpg	litter	375.5373	465.4305	391.4401	475.8993
58.jpg	litter	474.2009	535.0185	496.226	552.5499
59.jpg	litter	398.4356	60.69993	410.4574	83.04239
63.jpg	litter	294.5797	337.9469	312.997	351.6306
63.jpg	litter	342.892	281.2834	357.2458	300.8115
63.jpg	litter	476.8109	496.6656	520.7018	543.0143
63.jpg	litter	505.8934	749.5571	519.4775	761.3711
7.jpg	litter	213.5916	234.471	252.7215	280.4399
7.jpg	litter	214.9874	357.3376	234.6445	371.0321
7.jpg	litter	266.8785	336.1235	282.8512	355.8246
7.jpg	litter	300.1398	440.2656	343.0324	503.6425
7.jpg	litter	311.5909	437.408	343.2695	466.9063
7.jpg	litter	318.3254	456.4602	353.6767	484.2466
7.jpg	litter	366.731	605.8939	411.1686	679.0421
7.jpg	litter	414.0727	731.807	453.7668	785.2113
7.jpg	litter	562.6923	153.2562	573.669	162.846
71.jpg	litter	75.59468	12.89251	84.211	18.92631
71.jpg	litter	221.9807	322.9049	253.7473	349.4376
72.jpg	litter	274.5155	591.1003	301.6909	610.4728
72.jpg	litter	276.772	72.88114	299.7468	92.72649
72.jpg	litter	277.9052	337.7073	293.1284	359.2391
72.jpg	litter	283.457	314.2255	297.5203	328.259
72.jpg	litter	287.7047	619.642	308.4452	648.8243
72.jpg	litter	306.2287	515.9749	326.8889	528.7042
72.jpg	litter	322.3428	358.8549	347.1518	379.8506
72.jpg	litter	325.3402	725.4682	337.8728	738.0692
72.jpg	litter	331.2759	408.1158	350.4215	424.3917
72.jpg	litter	334.4681	701.3371	344.8338	715.4238
72.jpg	litter	338.471	450.0991	353.4849	472.0714

72.jpg	litter	354.3636	678.7099	363.4783	690.7205
72.jpg	litter	412.7406	495.1445	423.1526	509.5735
73.jpg	litter	179.909	748.1642	191.1627	761.3578
73.jpg	litter	215.5208	482.7548	232.3599	498.1095
73.jpg	litter	228.2373	439.3384	245.1453	458.0912
73.jpg	litter	258.6674	538.9193	294.3472	592.3035
73.jpg	litter	290.5826	472.8909	303.9309	487.1786
73.jpg	litter	308.0589	632.2907	321.8907	645.4239
73.jpg	litter	350.4119	698.0211	366.0836	714.0538
73.jpg	litter	367.6519	497.4426	382.7356	514.9009
73.jpg	litter	390.1156	451.4547	402.1254	462.1424
73.jpg	litter	400.6275	377.2516	411.2249	388.5864
73.jpg	litter	483.7384	443.9522	497.3225	457.5186
73.jpg	litter	484.0183	60.0301	497.7357	81.36383
73.jpg	litter	516.1474	500.2925	527.1298	512.9476
73.jpg	litter	525.4666	705.3051	539.4187	717.3669
74.jpg	litter	173.0398	279.7133	184.2387	289.6463
74.jpg	litter	230.4333	472.1514	249.8786	497.5284
74.jpg	litter	270.6924	753.6716	287.6474	768.9726
74.jpg	litter	281.4538	294.1744	292.6479	306.4222
74.jpg	litter	290.2193	205.6209	301.3518	218.605
74.jpg	litter	300.7339	691.4313	328.9481	727.9008
74.jpg	litter	308.5684	141.3818	324.6726	157.6544
74.jpg	litter	315.4424	562.4166	331.4175	585.4106
74.jpg	litter	316.1276	448.6057	329.4151	467.4161
74.jpg	litter	316.3301	515.0623	331.888	544.4665
74.jpg	litter	321.8707	417.8059	342.4912	433.8512
74.jpg	litter	331.2034	171.1696	342.5841	183.9123
74.jpg	litter	332.1277	196.8453	344.2758	210.3019
74.jpg	litter	337.7758	399.1376	351.4714	407.5256
74.jpg	litter	369.3739	428.1452	388.427	441.9943

75.jpg	litter	75.61849	384.3739	83.7103	392.2716
75.jpg	litter	189.9102	111.5318	198.4421	122.635
75.jpg	litter	268.732	240.894	277.4747	249.9653
75.jpg	litter	317.3053	336.8334	331.6998	349.0235
75.jpg	litter	319.9997	703.2314	339.1594	716.982
75.jpg	litter	338.3415	46.17987	351.0546	56.48593
75.jpg	litter	354.759	155.8413	368.46	169.8002
75.jpg	litter	358.361	262.0451	370.44	278.323
75.jpg	litter	377.3922	511.3139	389.8818	530.1192
76.jpg	litter	291.5431	520.5747	303.5899	530.3263
77.jpg	litter	381.1879	587.5256	389.717	599.0736
78.jpg	litter	412.8522	261.5911	425.0247	268.2892
80.jpg	litter	68.41406	214.423	79.40582	229.7129
80.jpg	litter	112.8958	172.6014	122.751	179.4256
80.jpg	litter	201.2949	676.1351	213.2899	689.2032
80.jpg	litter	233.932	707.6256	246.0602	717.542
80.jpg	litter	364.6357	625.0353	378.7306	649.6036
80.jpg	litter	409.6095	707.4406	422.4435	719.383
80.jpg	litter	492.5261	759.7844	511.2623	776.5877
81.jpg	litter	200.0586	505.8928	211.6135	515.5793
81.jpg	litter	201.151	601.6624	212.4136	609.6431
81.jpg	litter	367.5723	533.1318	377.5985	539.2401
81.jpg	litter	371.2298	334.8198	383.9686	343.8416
81.jpg	litter	388.7877	60.08981	412.9926	102.4622
81.jpg	litter	433.7624	175.7358	444.5926	186.7737
83.jpg	litter	58.13385	559.2142	67.92766	570.2934
83.jpg	litter	75.72945	644.2544	85.60925	654.1794
83.jpg	litter	351.3578	488.8926	363.3466	503.0444
83.jpg	litter	353.9035	395.8553	463.5765	523.7252
84.jpg	litter	121.0588	518.5868	133.8538	527.0239
84.jpg	litter	156.0204	468.4182	172.4309	486.5457

84.jpg	litter	189.0913	290.573	340.3146	491.0056
84.jpg	litter	479.6986	261.5562	539.9544	343.2596
85.jpg	litter	233.1744	373.4784	248.9214	387.0655
85.jpg	litter	236.6359	303.8642	245.3536	315.783
85.jpg	litter	258.0599	496.3651	269.8987	509.1987
85.jpg	litter	274.3372	642.2713	283.7905	654.1701
85.jpg	litter	286.2322	570.5073	299.3637	584.8112
85.jpg	litter	391.5011	99.60356	399.976	114.0053
85.jpg	litter	419.4835	504.6086	430.5473	514.0692
85.jpg	litter	430.2027	568.3313	447.2371	580.6905
85.jpg	litter	459.6051	273.4107	474.1788	284.8827
85.jpg	litter	461.1206	308.6167	475.3212	322.9559
85.jpg	litter	481.9099	504.2784	493.3071	518.0166
85.jpg	litter	487.5775	409.0633	522.4273	473.942
85.jpg	litter	515.0544	602.7614	529.3822	618.7047
85.jpg	litter	516.7161	506.6331	529.058	520.9403
85.jpg	litter	524.3674	570.1839	538.118	585.3769
86.jpg	litter	122.0376	296.9244	131.633	304.7202
86.jpg	litter	151.2298	496.5051	165.5138	508.7612
86.jpg	litter	169.3995	311.2232	181.9856	331.9917
86.jpg	litter	225.9329	330.0749	241.7483	349.2697
86.jpg	litter	240.7677	184.3561	249.9424	195.2382
86.jpg	litter	300.581	366.765	328.5515	410.9309
86.jpg	litter	318.8858	596.535	335.1041	610.6476
86.jpg	litter	319.7596	562.6336	329.7457	576.5093
86.jpg	litter	327.9694	538.9792	339.7726	551.9148
86.jpg	litter	351.8206	710.779	359.8895	721.5742
86.jpg	litter	376.7268	546.1524	389.2454	567.0896
86.jpg	litter	383.1903	391.1652	398.0763	412.2991
88.jpg	litter	179.2325	231.5861	196.8432	247.6177
88.jpg	litter	246.6062	240.7592	263.3129	262.7026

88.jpg	litter	292.9978	430.3595	304.6022	438.439
88.jpg	litter	566.4062	442.9193	575.7252	456.4139
9.jpg	litter	451.1481	240.3068	465.4045	255.5687
91.jpg	litter	408.0283	442.2748	421.7258	456.2821
97.jpg	litter	218.1049	467.8305	227.5663	479.2587
97.jpg	litter	223.023	542.0363	232.4819	550.9301
97.jpg	litter	228.7339	231.9929	235.8645	240.484
97.jpg	litter	237.8597	386.5605	247.4208	399.6117

Appendix E: Python Code to Construct a Model

The code can be seen in Google Colaboratory via this link: https://drive.google.com/file/d/1w-aJknKci6dSlQgc_HVPCXrWDKpFPWS4/view?usp=sharing

```
# mount with your google drive if you want to use the data from them
# restart runtime after installed

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

!pip install tensorflow-gpu==1.15.0 --force-reinstall

#check Tensorflow version (must be 1.x)
import tensorflow
print(tensorflow.__version__)
```

Setting the parameters and choosing the model you want to use

```
# Number of training steps.
num_steps = 2000  # 200000

# Number of evaluation steps.
num_eval_steps = 50

MODELS_CONFIG = {
    'faster_rcnn_resnet101_coco': {
        'model_name': 'faster_rcnn_resnet101_coco_2018_01_28',
        'pipeline_file': 'faster_rcnn_resnet101_coco.config',
        'batch_size':1
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
        'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
        'batch_size': 1
    },
    'rfcn_resnet101': {
        'model_name': 'rfcn_resnet101_coco_2018_01_28',
        'pipeline_file': 'rfcn_resnet101_pets.config',
```

```
        'batch_size': 1
    }
}

# Pick the model you want to use
# Select a model in `MODELS_CONFIG`.
selected_model = 'faster_rcnn_inception_v2'

# Name of the object detection model to use.
MODEL = MODELS_CONFIG[selected_model]['model_name']

# Name of the pipline file in tensorflow object detection API.
pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']

# Training batch size fits in Colabe's Tesla K80 GPU memory for selected mode
l.
batch_size = MODELS_CONFIG[selected_model]['batch_size']

# clone Tensorflow API
%cd /content/

!git clone --quiet https://github.com/tensorflow/models.git

#%cd /content/drive/My Drive/scrapbook_object_detection

!ls

# install Protocol Buffers and Cython for managing protocol buffers and its e
nvironment
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib

!pip install -q pycocotools

%cd /content/models/research
!protoc object_detection/protos/*.proto --python_out=.
import os
```

```
os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/resea
rch/slim/'

# When start a new session or reopen, the code below needs to be run when usi
ng google drive
# Change path
#%cd /content/drive/My Drive/scrapbook_object_detection/models/research
#!protoc object_detection/protos/*.proto --python_out=.
#import os
#os.environ['PYTHONPATH'] += ':/content/drive/My Drive/scrapbook_object_detec
tion/models/research/:/content/drive/My Drive/scrapbook_object_detection/mode
ls/research/slim/'

# install tf_slim
!pip install tf_slim

# # check if all packages were installed yet?
!python object_detection/builders/model_builder_test.py
```

Upload TF Records File

- You are able to use you dataset by chaging the file name to your own file in this section
- There are three files that you have to create, "test.record", "train.record", and "label.pbtxt" files.

```
%cd /content/
!wget https://www.dropbox.com/s/2mmwf59v3qoi3y0/dataset.zip

#unzip file
!unzip dataset.zip

test_record_fname = '/content/dataset/data/test.record'
train_record_fname = '/content/dataset/data/train.record'
label_map_pbtxt_fname = '/content/dataset/training/label_map.pbtxt'

%cd /content/models/research
```

```
import os
import shutil
import glob
import urllib.request
import tarfile
MODEL_FILE = MODEL + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
DEST_DIR = '/content/models/research/pretrained_model'

if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
    shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)

!echo {DEST_DIR}
!ls -alh {DEST_DIR}

fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
fine_tune_checkpoint
```

Configuring a Training Pipeline

```
import os
pipeline_fname = os.path.join('/content/models/research/object_detection/samp
les/configs/', pipeline_file)

assert os.path.isfile(pipeline_fname), '`{}` not exist'.format(pipeline_fname
)

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
```

```
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())
import re

num_classes = get_num_classes(label_map_pbtxt_fname)
with open(pipeline_fname) as f:
    s = f.read()
with open(pipeline_fname, 'w') as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # tfrecord files train and test.
    s = re.sub(
        '(input_path: ".*?)(train.record)(.*?")', 'input_path: "{}"'.format(t
rain_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(val.record)(.*?")', 'input_path: "{}"'.format(tes
t_record_fname), s)

    # label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtx
t_fname), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
```

```
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)
    f.write(s)
!cat {pipeline_fname}
#model_dir = 'training/'
# Optionally remove content from previous output model directory to fresh sta
rt.
# you can create new training folder by changing to i.e. training_model1
#!rm -rf {model_dir}
#os.makedirs(model_dir, exist_ok=True)
```

Training the model using code below

```
!python /content/models/research/object_detection/legacy/train.py \
    --pipeline_config_path={pipeline_fname} \
    --train_dir='/content/dataset/training' \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

Exporting a Trained Inference Graph

- Once your training job is complete, you need to extract the newly trained inference graph, which will be later used to perform the object detection.
- you can change directory to your path

```
# You need to change the training number after "model.ckpt-XXXX" by look a ch
eckpoint in training folder
%cd '/content/models/research/object_detection'
!python export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path '/content/dataset/training/pipeline.config' \
    --trained_checkpoint_prefix '/content/dataset/training/model.ckpt-2000' \
    --output_directory '/content/dataset/my_model'

!zip -r '/content/dataset/my_model.zip' '/content/dataset/my_model'
```

Tensorboard

```
# Load the TensorBoard notebook extension
```

```
# Input evaluation or training path
%load_ext tensorboard
tensorboard --logdir '/content/dataset/training'
```

Confusion Matrix for each object

```
%cd '/content/models/research/object_detection/inference'
!python infer_detections.py \
    --input_tfrecord_paths='/content/dataset/data/test.record' \
    --output_tfrecord_path='/content/dataset/detections.tfrecord' \
    --inference_graph='/content/dataset/my_model/frozen_inference_graph.pb'
# clone confusion matrix for object detection

%cd /content/
!git clone --quiet https://github.com/svpino/tf_object_detection_cm.git

%cd /content/tf_object_detection_cm
!python confusion_matrix.py \
    --detections_record='/content/dataset/detections.tfrecord' \
    --label_map='/content/dataset/training/label_map.pbtxt' \
    --output_path='/content/dataset/confusion_matrix.csv'
```

Model Evaluation

- It has to be run in parallel with the object detection training notebook
- Use to compute mAP score

```
%cd '/content/drive/My Drive/scrapbook_object_detection/models/research/objec
t_detection/legacy'
!python eval.py \
    --logtostderr \
    --pipeline_config_path='/content/drive/My Drive/scrapbook_object_detectio
n/6_classes/case3_476/training_inception_100000/pipeline.config' \
    --checkpoint_dir='/content/drive/My Drive/scrapbook_object_detection/6_cl
asses/case3_476/training_inception_100000' \
    --eval_dir='/content/drive/My Drive/scrapbook_object_detection/6_classes/
case3_476/eval_inception_100000'/
```

Appendix F: Python Code for Prediction

The code can be seen in Google Colaboratory via this link:
https://drive.google.com/file/d/1uNwUNZd0hyre4G77SbQ0Xfq8RxsARBnZ/view?usp=sharing

```
# mount with your google drive if you want to use the data from them
from google.colab import drive
drive.mount('/content/drive')


# install the tensorflow gpu version 1.15.0 because the pretained models that
we use is for with version 1.15.0
# restart runtime after installed
!pip install tensorflow-gpu==1.15.0 --force-reinstall


#check Tensorflow version (must be 1.x)
import tensorflow
print(tensorflow.__version__)


# clone Tensorflow API
%cd /content/

!git clone --quiet https://github.com/tensorflow/models.git


# install Protocol Buffers and Cython for managing protocol buffers and its e
nvironment
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib

!pip install -q pycocotools


%cd /content/models/research
!protoc object_detection/protos/*.proto --python_out=.
import os
os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/resea
rch/slim/'
```

```
# install tf_slim
!pip install tf_slim

# Downdload trained model
!wget https://www.dropbox.com/s/vht5rxijg4jj3rs/model.zip

# Downdload testing data
!wget https://www.dropbox.com/s/mknz89biq913p0w/test_images.zip

#unzip file
!unzip model.zip


!unzip test_images.zip

# change the directory to your own part
PATH_TO_FOLDER = '/content'


!ls

MODELS_PATH = {
    'model8_faster_rcnn_resnet101': {
        'model_path': '/content/model/model8_resnet101/model_resnet101/frozen
_inference_graph.pb'
    },
    'model7_faster_rcnn_inception_v2': {
        'model_path': '/content/model/model7_inception/model_inception_100000
/frozen_inference_graph.pb'
    },
    'model9_faster_rcnn_inception_resnet_v2': {
        'model_path': '/content/model/model9_inception_resnet/model_inception
_resnet1/frozen_inference_graph.pb'
    }
}

# Pick the model you want to use
```

```
# Select a model in `MODELS_CONFIG`.
selected_model = 'model8_faster_rcnn_resnet101'

# Name of the object detection model to use.
TRAINED_MODEL_PATH = MODELS_PATH[selected_model]['model_path']

# import libraries and set up the model
import os
import cv2
import numpy as np
import tensorflow as tf
import pandas as pd
import sys
%matplotlib inline
%matplotlib notebook
sys.path.append(PATH_TO_FOLDER)
sys.path.append(os.path.join(PATH_TO_FOLDER,"models/research"))
sys.path.append(os.path.join(PATH_TO_FOLDER,"models/research/object_detection
/utils"))

import sys
sys.path.append(os.path.join(PATH_TO_FOLDER))
PATH_TO_CKPT = os.path.join(TRAINED_MODEL_PATH)
PATH_TO_LABELS = os.path.join('/content/model/label_map.pbtxt')
from object_detection.utils import label_map_util
import visualization_utils as vis_util
from google.colab.patches import cv2_imshow

def predict(PATH_TO_IMAGE):

  NUM_CLASSES = 6

  label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
  categories = label_map_util.convert_label_map_to_categories(label_map, max_
num_classes=NUM_CLASSES, use_display_name=True)
  category_index = label_map_util.create_category_index(categories)
```

```
  detection_graph = tf.Graph()
  with detection_graph.as_default():
      od_graph_def = tf.GraphDef()
      with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
          serialized_graph = fid.read()
          od_graph_def.ParseFromString(serialized_graph)
          tf.import_graph_def(od_graph_def, name='')

      sess = tf.Session(graph=detection_graph)

  image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')


  detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')


  detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
  detection_classes = detection_graph.get_tensor_by_name('detection_classes:0
')

  num_detections = detection_graph.get_tensor_by_name('num_detections:0')

  # read image
  image_path = PATH_TO_IMAGE
  image = cv2.imread(PATH_TO_IMAGE)
  image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image_expanded = np.expand_dims(image_rgb, axis=0)

  # get dimensions of image
  dimensions = image.shape

  # height, width, number of channels in image
  height = image.shape[0]
  width = image.shape[1]
  channels = image.shape[2]

# Actual detection
  (boxes, scores, classes, num) = sess.run(
```

```
      [detection_boxes, detection_scores, detection_classes, num_detections],
      feed_dict={image_tensor: image_expanded})

  boxes = np.squeeze(boxes)
  scores = np.squeeze(scores)
  classes = np.squeeze(classes).astype(np.int32)

#Show only one class, i.e if you want only litter class input classes == 1, t
he number comes from label map
  #indices = np.argwhere(classes == 1)
  #boxes = np.squeeze(boxes[indices])
  #scores = np.squeeze(scores[indices])
  #classes = np.squeeze(classes[indices]).astype(np.int32)

# Visualisation of the results of a detection
  vis_util.visualize_boxes_and_labels_on_image_array(
      image,
      boxes,
      classes,
      scores,
      category_index,
      use_normalized_coordinates=True,
      line_thickness=2,
      min_score_thresh=0.75)

  H,W,_ = image.shape
  im = cv2.resize(image, (int(W),int(H)))
  count = 0
  for score in np.squeeze(scores) :
    if score >= 0.75 : count += 1
    else : None

# display image and print the scores and the location of predicted boxes
  cv2_imshow(im)
  for i, box in enumerate(np.squeeze(boxes)):
#for i in range(0,count) :
    class_name = category_index[np.squeeze(classes)[i].astype(np.int32)]['nam
e']
```

```
    if np.squeeze(scores)[i] >= 0.75 :
      print(class_name,'\t',np.squeeze(scores)[i])
      print("ymin={}, xmin={}, ymax={}, xmax{}".format(box[0]*height,box[1]*w
idth,box[2]*height,box[3]*width))
    else : None

# print image filename
  print (os.path.split(image_path)[-1])

# Store results in predicted_data
  predicted_data = []
  new_class_name = []
  new_box0 = []
  new_box1 = []
  new_box2 = []
  new_box3 = []
  new_image_name = []
  for i, box in enumerate(np.squeeze(boxes)):
      class_name = category_index[np.squeeze(classes)[i].astype(np.int32)]['n
ame']
      image_path = os.path.split(image_path)[-1]
      box0 = box[0]*height
      box1 = box[1]*width
      box2 = box[2]*height
      box3 = box[3]*width
      if (np.squeeze(scores)[i] > 0.75):
          new_image_name.append(image_path)
          new_class_name.append(class_name)
          new_box0.append(box0)
          new_box1.append(box1)
          new_box2.append(box2)
          new_box3.append(box3)

  predicted_data = list(zip(new_image_name,new_class_name,new_box0,new_box1,n
ew_box2,new_box3))

# create dataframe
```

```
  df = pd.DataFrame (predicted_data, columns = ['image_name','class','ymin','
xmin','ymax','xmax'])
# save dataframe to CSV file
  df.to_csv('csvfile1.csv', index = False)

from os import listdir
from os.path import join

def predict_dir(PATH_TO_DIR) :


  NUM_CLASSES = 6

  label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
  categories = label_map_util.convert_label_map_to_categories(label_map, max_
num_classes=NUM_CLASSES, use_display_name=True)
  category_index = label_map_util.create_category_index(categories)

  detection_graph = tf.Graph()
  with detection_graph.as_default():
      od_graph_def = tf.GraphDef()
      with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
          serialized_graph = fid.read()
          od_graph_def.ParseFromString(serialized_graph)
          tf.import_graph_def(od_graph_def, name='')

      sess = tf.Session(graph=detection_graph)

  image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')


  detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')


  detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
  detection_classes = detection_graph.get_tensor_by_name('detection_classes:0
')
```

```
  num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# create variables
  predicted_data = []
  new_class_name = []
  new_box0 = []
  new_box1 = []
  new_box2 = []
  new_box3 = []
  new_image_name = []

  filename = listdir(PATH_TO_DIR)
  for i in range(0,len(filename)) :
    if filename[i].endswith('.jpg') |  filename[i].endswith('.png')|filename[
i].endswith('.jpeg') :

  # read image
      PATH_TO_IMAGE = join(PATH_TO_DIR,filename[i])
      image_path = PATH_TO_IMAGE
      image = cv2.imread(PATH_TO_IMAGE)
      image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
      image_expanded = np.expand_dims(image_rgb, axis=0)

  # get dimensions of image
      dimensions = image.shape

  # height, width, number of channels in image
      height = image.shape[0]
      width = image.shape[1]
      channels = image.shape[2]

# Actual detection
      (boxes, scores, classes, num) = sess.run(
          [detection_boxes, detection_scores, detection_classes, num_detectio
ns],
          feed_dict={image_tensor: image_expanded})

      boxes = np.squeeze(boxes)
```

```
      scores = np.squeeze(scores)
      classes = np.squeeze(classes).astype(np.int32)

#Show only one class, i.e if you want only litter class input classes == 1, t
he number comes from label map
      #indices = np.argwhere(classes == 1)
      #boxes = np.squeeze(boxes[indices])
      #scores = np.squeeze(scores[indices])
      #classes = np.squeeze(classes[indices]).astype(np.int32)

# Visualisation of the results of a detection
      vis_util.visualize_boxes_and_labels_on_image_array(
          image,
          boxes,
          classes,
          scores,
          category_index,
          use_normalized_coordinates=True,
          line_thickness=2,
          min_score_thresh=0.75)

      H,W,_ = image.shape
      im = cv2.resize(image, (int(W),int(H)))
      count = 0
      for score in np.squeeze(scores) :
        if score >= 0.75 : count += 1
        else : None

# display image and print the scores and the location of predicted boxes
      cv2_imshow(im)
      print (os.path.split(image_path)[-1]) # print image filename
      for i, box in enumerate(np.squeeze(boxes)):
#for i in range(0,count) :
        class_name = category_index[np.squeeze(classes)[i].astype(np.int32)][
'name']
        if np.squeeze(scores)[i] >= 0.75 :
          print(class_name,'\t',np.squeeze(scores)[i])
```

```
          print("ymin={}, xmin={}, ymax={}, xmax{}".format(box[0]*height,box[
1]*width,box[2]*height,box[3]*width))
        else : None



# Store results in predicted_data
      for i, box in enumerate(np.squeeze(boxes)):
          class_name = category_index[np.squeeze(classes)[i].astype(np.int32)
][['name']
          image_path = os.path.split(image_path)[-1]
          box0 = box[0]*height
          box1 = box[1]*width
          box2 = box[2]*height
          box3 = box[3]*width
          if (np.squeeze(scores)[i] > 0.75):
              new_image_name.append(image_path)
              new_class_name.append(class_name)
              new_box0.append(box0)
              new_box1.append(box1)
              new_box2.append(box2)
              new_box3.append(box3)

  predicted_data = list(zip(new_image_name,new_class_name,new_box0,new_box1,n
ew_box2,new_box3))

# create dataframe
  df = pd.DataFrame (predicted_data, columns = ['image_name','class','ymin','
xmin','ymax','xmax'])
# save dataframe to CSV file
  df.to_csv('result_inception_100000.csv', index = False)
```

predict using 1 picture

```
predict('/content/test_images/29.jpg')
```

predict all images in folder

```
predict_dir('/content/test_images')
```