Eye in the Sky: Object Detection and Tracking via Drone Imagery using Convolutional Neural Networks

Student Name: Mr. Matthew Ian Boyle
Supervisor Name: Dr. Yona Falinie Binti-Abd-Gaus
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—In this paper we evaluate how well state-of-the-art (SOTA) computer vision algorithms apply to drone imagery. We first look at a SOTA one-stage detection algorithm and a SOTA two-stage detection algorithm and compare their performance in this domain. We also look at recent extensions to standard state-of-the-art object detection algorithms to show how they can improve performance in this domain by addressing particular issues associated with it, i.e. the effects of capturing footage at higher altitudes. We then look at how different object tracking algorithms perform on the same domain when using each of the aforementioned object detection algorithms as input. Overall, we find that YOLOv3, the standard one-stage SOTA model, is the best object detection candidate for real-time applications, out of the object detection models experimented with, due to its significantly faster processing speeds. However, we see that for tasks where predictive performance is more important than processing speed, an enhancement of Faster R-CNN which utilises a CARAFE-based FPN is the best solution when compared to all other models experimented with. We also see that the BYTETrack algorithm is a suitable candidate for object tracking in this domain in both real-time applications and applications which allow for processing at a later time.

Index Terms—Object Detection, Object Tracking, Convolutional Neural Networks, UAV imagery

1 Introduction

Object detection is a classic problem in computer vision, which aims to locate instances of objects within an image or frame of a video. This is an area of study that has been extensively researched over the last two decades, however, still provides many challenges to researchers in the field. The general approach to object detection is to take a single image (or frame from video) and predict the probability that a pre-defined object (e.g. a car) lies within a specific region of the image. This is region is defined by a bounding box which outlines the expected location of the object within the image.

Object detection can be further sub-divided into two categories: Single Object Detection and Multiple Object Detection. As the names suggest, Single Object Detection aims to detect the presence of a single object within an image, whereas Multiple Object Detection aims to be able to detect multiple objects, potentially of multiple different classes, within an image. Multiple Object Detection, by nature, is considered more complex than Single Object Detection, as there can be potentially many distinct objects the model needs to be able to localise. Furthermore, there can be multiple instances of different classes with in the frame that need to be correctly distinguished.

The majority of algorithms used to perform object detection are built using Convolutional Neural Networks (CNNs) and can generally be split into two categories: One-Stage detectors [1] [2] [3] and Two-Stage detectors [4] [5]. In their approach to object detection, two-stage detectors work by using two sub-models, the first of which proposes regions where objects may be present and the second which aims to classify the proposed objects and further refine the accuracy of the generated bounding box. On the other hand, one-stage detectors work by regressing to the bounding box in a single step, making use of a grid-box and anchors to find the object within the image. Previous research [6] has shown that one-stage detectors allow for faster inference than two-stage detectors, whilst two-stage detectors boast a higher recognition accuracy. This makes sense given the higher complexity of the architectures of two-stage detectors.

Object tracking is an extension of the object detection problem which aims to track an object detected in a frame of video across multiple frames. This requires an algorithm to assign each detected object a unique identifier and then if the same object appears in a subsequent frame, assign the same identifier to the object in that frame. In a similar manner to object detection, object tracking can be broken down into single object tracking and multiple object tracking. Multiple object tracking is inherently more difficult to solve than single object tracking as not only does an algorithm have to be able to consistently detect multiple objects in multiple frames, but it must also be intelligent enough not to confuse different objects of the same class for one another. For example, if there are two cars in a frame labelled car_1 and car_2, in the next frame they should retain their same labels, i.e. car_1 should not be mistaken for car_2 and vice versa.

The general approach taken for object tracking is to take a pre-trained object detection model and use the detector as

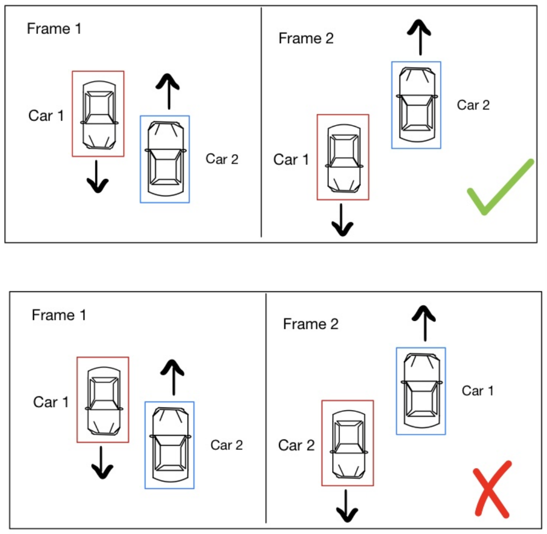

Fig. 1. Depiction of how multiple object tracking should appear across frames.

input into an algorithm which assigns identifiers to objects based on their appearance and/or behaviours pertaining to how they move between frames, such that they can correctly identify them in the next frame.

Whilst there has been an extensive amount of research into the development and testing of object detection and tracking methods within domains such as autonomous vehicles, there has not yet been a great level of research on their application into drone imagery. Drones are Unmanned Aerial Vehicles (UAVs) which have seen use in a variety of sectors, including leisure, surveillance, aerial photography and search and rescue. The market for drone related services is expected to grow massively in the near future [7].

The use of object detection/tracking software in camera equipped drones can lead to automation within a variety of industrial practices. For example, from a security perspective, such software could allow for the surveillance of restricted areas/private property to automatically detect intruders/security breaches.

The agricultural sector has also seen the potential for object tracking drones as recently, farmers in New Zealand have started using drones to herd sheep. Yaxley et al. [8] shows that using these drones has had a notable beneficial impact on the welfare of sheep by reducing their stress levels when compared to traditional herding methods involving sheep dogs. Their use also reduces the risk to farmers. Incorporating the use of object tracking software could aid in the development of automated sheep herding via drones which will help to reduce farmers workloads.

The COVID-19 lockdowns imposed in the UK across 2020 and 2021 saw a rise in the use of food delivery services [9]. The use of automated delivery drones could help to improve the overall efficiency of the delivery industry by reducing delivery times, as the drone's ability to fly would eliminate delays caused by traffic. Furthermore, with the introduction of 'contact free' delivery during the pandemic, use of delivery drones could also play a role in keeping customers safe going forward, as it minimises contact with other humans throughout the delivery process. The automation of drone-based delivery would require object detection and tracking software to be used in order to avoid obstacles that the drone may otherwise crash into.

There is also a potential application within aerial photography, where the ability of a drone-based camera to detect the intended target's location within a frame can allow for automatic adjustment of camera settings, such as aperture, shutter speed and ISO levels in order to take the best photo possible.

Whilst video footage captured from UAVs maintains many classic challenges for object detection and tracking, such as occlusions and varying weather conditions, it also presents some of its own more unique challenges. Such challenges are caused by the camera motion and altitude variation caused by the flying patterns of the drone as well as the fact that the objects present in the footage can often be very small and high in quantity, due to the altitude of the drone when the images are taken and the ”regular use of wide angle lenses” [10]. These features can make it difficult for many standard state-of-the-art object detection and tracking algorithms to achieve high performance on drone-based imagery. Therefore, it is necessary to look beyond the standard state of the art object detection/tracking methods when performing object detection and look into using algorithms that more closely address the particular problems presented by drone footage.

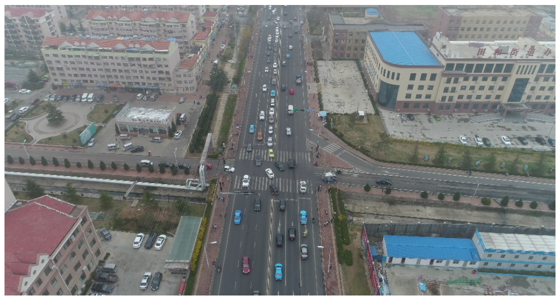

Fig. 2. Frame captured from [11], showing high object density and small objects as a result of high flying altitude.

This project aims to answer the question: *how well do state-of-the-art computer algorithms apply to the drone imagery domain and what methods can be used to gain improved performance?* To achieve this, we train and then compare the performance of existing SOTA object detection and tracking algorithms when applied to the drone imagery domain, specifically focusing on detecting and tracking moving vehicles for surveillance purposes. We also look to test recent extensions to SOTA algorithms that we expect should enhance them such that they better tackle the issues specifically presented by detection/tracking in drone based imagery. The objectives of the project can be split into three sections as follows.

Basic Objective: To investigate two end-to-end CNN object detection architechture, which difer in design, i.e one-stage vs two-stage from drone based imagery by balancing processing time and performance in real time detection.

Intermediate Objective: To examine the impact of two additional object detection algorithms chosen towards specific issues in drone based imagery such as high altitude and high-object density.

Advanced Objective: To adopt an existing CNN object detection algorithm chosen, by associating every detection box with tracking for each and every frame under drone based imagery label.

2 Related Work

2.1 Classical Object Detection

2.1.1 Convolutional Neural Networks

As mentioned in the Introduction, the majority of object detection algorithms are built upon the use of Convolution Neural Networks (CNNs) [12]. These networks consist of a combination of Convolutional, Pooling and sometimes Normalisation layers and allow for the network to learn the spatial and temporal dependencies of an image which are normally lost when using the likes of Multi-Layered Perceptrons.

There are many different CNN architecturs, which have different characteristics that make them useful for different detection algorithms. One of the most popular CNNs used in object detection is the ResNet architecture [13] which utilises skip connections to enforce the idea that deep layers don't have more training mistakes than shallower ones. These skip-connections also minimise the vanishing gradient problem as gradients are retained through the shortcut link from earlier layers. The ResNeXt architecture [14] expands upon ResNet by adding parallel layers with shared hyper-parameters within the residual blocks. These parallel layers are then merged together before merging the output of the block with that of the skip-connection. Both ResNet and ResNeXt are used as backbones of many common object detection methods and commonly contain either 50 or 101 layers, chosen depending on the need for speed vs performance respectively.

2.1.2 One-Stage Detectors

One of the most famous One-Stage detection algorithms which makes use of CNNs and is currently considered state-of-the-art is YOLOv3 [1] developed to provide fast training and inference times. The algorithm approaches object detection as a regression problem by using logistic regression to predict confidence that the an object lies within a certain area. This is done within an $S \times S$ grid whereby each cell is responsible for the detection of a single object.

Single-Shot Multi-Box Detector (SSD) [2] is a One-Stage detection algorithm, which like YOLOv3, splits the input image into a grid, but instead allows for the detection of multiple objects within a single grid cell. The model works by using a pre-trained classification network as a backbone and an 'SSD Head' which contain additional convolutional layers which predict the bounding boxes for each object.

Retina-Net [3] is another One-Stage detector developed to address the 'extreme foreground-background class imbalance' that the authors claim is the reason that One-Stage detectors are generally less accurate than their Two-Stage counterparts. They attempt this by adjusting cross-entropy loss to down-weight the loss assigned to well classified examples. The authors found that their algorithm maintained the fast training and inference characteristics of the other One-Stage detectors whilst surpassing the accuracy of all known Two-Stage detectors at the time of its release.

The FreeAnchor [15] method was proposed in 2019 which modifies the way that One-Stage detectors are evaluated so that they can better detect slender objects and objects that are clustered together. They re-frame the object detection problem as a Maximum-Likelihood Estimation procedure in order to update the anchor matching process such that the model learns how to anchors match to ground truths as opposed to matching the empirically. This allows for objects to be matched to boxes more flexibly by breaking the restriction of the Intersect-Over-Union constraint that is usually used to train the detectors. The authors designed this method to be compatible with any One-Stage object detection algorithm.

2.1.3 Two-Stage Detectors

Faster R-CNN [4] is the most commonly used Two-Stage detection algorithm and like YOLOv3 is considered to be state-of-the-art. Unlike YOLOv3, which was designed to provide high inference speed, Faster R-CNN prioritises detection accuracy at the expense of being a bit slower. The first stage of the algorithm uses a Region Proposal Network (RPN) to suggest Regions of Interest (ROIs) in the image where an object may be present. The second stage comprises of a pooling layer into which these ROIs are fed, followed by a fully connected layer which has a regression layer to predict the bounding boxes of the objects and a soft-max layer to classify the object.

Many implementations of Faster R-CNN incorporate Feature Pyramid Networks (FPN) [16] to improve the quality of the features fed into the RPN in the first stage. When training an object detection model, it is possible to use pyramids of the same image at different scales were used to improve the algorithms ability to detect small images. This, however, is very inefficient as it requires a lot of memory and a lot of training time. FPNs instead use a pyramid of feature maps which contain more useful information than re-scaled images. The FPN consists of two components, the first being a bottom-up pathway which uses a CNN which in each layer reduces the resolution of the image but improves the usefulness of the image by showing more high-level structures. The second component is a top-down pathway which takes the lower resolution outputs and attempts to reconstruct higher resolution equivalents of them, utilising skip-connections and lateral connections from the layers in the bottom-up component. These two components are used to generate the pyramid of features.

Another commonly used Two-Stage detector is Cascade R-CNN [5], which contains a sequence of detectors, which are designed based on the second stage of Faster R-RCNN, trained using higher IoU thresholds as the sequence progresses, and feeding gradients calculated by each detector into the subsequent one. This method reduces the level of over-fitting observed during training and the improves the overall quality of detections.

The Content-Aware ReAssembly of FEatures (CARAFE) method [17], was developed to improve upon common feature up-sampling methods and can be used in two-stage detectors such as Faster R-CNN when applied to object detection. In Faster R-CNN, CARAFE replaces the use of nearest-neighbour interpolation in the top-down pathway of the FPN. The method itself uses weights generated in a content-aware manner to reassemble features of a region. This results in feature maps which more accurately, by upsampling them spatially and by enhancing their discrimination. The authors claim they saw an improvement in mAP scores when using CARAFE with Faster R-CNN compared to without it. The improvement in feature upscaling could further improve the ability of the detectors which utilise it to better detect small objects.

2.2 UAV Datasets

The majority of progression in drone-based computer vision has come as a result of the release of the VisDrone dataset [18]. The authors created a comprehensive drone-based object detection and tracking dataset consisting of 10k+ static images for object detection, 139k.3 frames for single object tracking and 40k frames for multiple object tracking, with *approx.* 183.3k average annotations per each of its class labels. As the authors intended for this dataset to be used as a bench mark for comparing object detection and tracking algorithms on drone footage, they also published a series of challenges alongside their dataset, encouraging people to try and solve one of 4 tasks; object detection in images, object detection in video, single object tracking in video and multiple object tracking in video. The authors annually publish solutions submitted to them along with a comprehensive performance analysis and comparison.

Mueller et al. [19] also aimed to create a benchmark dataset for testing object detection and tracking algorithms on drone footage. Their dataset contains 123 instances of HD video sequences captured from low-altitude drones. This dataset, containing over 110k frames, is annotated to show all instances of pedestrians and vehicles in each frame as well as containing attributes to describe the tracking challenges contained in each video (i.e. occlusions, scale variations, fast motion etc.).

Another significant dataset is UAVDT [11], which contains 80k frames and 841.5k annotations to learn from. The dataset contains 3 classes, specifically cars, buses and trucks and provides a range of conditions/scenarios on which detection and tracking can be tested. This is the dataset we plan to use for experimentation within this project. The creators of this dataset also provide an analysis of how some basic out-of-the box object detection algorithms perform on this dataset.

The DOTA [20] dataset aims to facilitate testing and training object detection algorithms on aerial imagery. The dataset contains 2,806 aerial images and approx. 188k annotations that can be used to train models to detect 15 class labels. Whilst not solely addressing drone imagery, aerial imagery contains many of the same challenges faced by drone imagery, particularly the inclusion of small objects (due to the high altitude of the camera). The creators of this dataset carried out a similar study as [11].

2.3 Drone-based Object Detection

The first release of results for the VisDrone Object detection challenges in 2018 [21] contained a large number of proposed solutions all with a varying degree of performance. The most standout solutions were the HAL-Retina-Net, DPNet, DE-FPN and CFE-SSDv2 algorithms. The HAL-Retina-Net algorithm [21] takes the principles of the RetinaNet algorithm and improves it for the recognition of low-resolution objects by removing P6 and P7 from the FPN in the feature extractor. It also includes Soft-NMS [22] and bounding box voting in the post-processing stage to improve accuracy. Further accuracy improvements are gained by increasing the normalised size of the input images. The RetinaNet model is trained using an SE-ResNeXt-50 [23] backbone. Of all the submissions to the 2018 VisDrone object detection in images competition, this algorithm boasted the best mAP score.

The DPNet [21] algorithm had the next best performance for this round of submissions and used an approach based on Faster-RCNN with FPN, training an ensemble of detectors using ResNet-50, ResNet-101 and ResNeXt backbones pre-trained on the ImageNet dataset [24]. Like HAL-Retina-Net they used Soft-NMS when predicting bounding boxes. They also used multi-scale training to improve results.

DE-FPN [21] approaches the object detection problem by using an FPN based approach. They advance upon the basic algorithm by enhancing the training data through image cropping and colour jitter. The developers trained this model using a ResNeXt-101 64-d backbone pretrained on the COCO dataset. Similarly to HAL-Retina-Net the authors adapted their models pyramid (by removing levels) in order to improve the models ability to detect low-resolution objects.

CFE-SSDv2 [21] differs drastically from the other algorithms mentions so far in that instead of being based on a two-stage detector, it builds upon SSD. The model performs 'Comprehensive Performance Enhancement' by enhancing the weak features in the model to improve the model's ability to detect small objects. Performance is further enhanced using a multi-scale inference. Model training is performed at 800x800 but inferencing uses a larger scale of 2200x2200.

The subsequent release of results for the VisDrone object detection challenge (2019) [25] provided another large selection of highly performing solutions. The highest performing from this iterations were DPNet-ensemble, RRNet and ARM-OC.

The DPNet-ensemble algorithm [25] uses a pair of detectors based on the Cascade-RCNN architecture and using FPN with one detector using a backbone of ResNet-50 and the other ResNet-101 both pre-trained on the ImageNet dataset [24]. These backbones are enhanced through the introduction of a global-context module and deformable convolution. The developers trained the model at multiple scales and used a balance strategy from Libra R-CNN [26] to ensure the model was capable of handling both large and small objects.

RRNet [25] takes a different approach by treating the objects as points, using a Convolutional Neural Network to predict the centre point and size of objects. These predictions are used to generate a course bounding box. These boxes are then processed through a Re-Regression and a ROIAlign module and further convolutional layers to refine the course bounding boxes. Soft-NMS [22] is then used to predict the final boxes.

The creators of ACM-OD [25] base their model on Faster-RCNN with FPN and a ResNet-101 backbone that has been pre-trained on MS COCO. To enhance their model for use on drone imagery, they use a process called Augmented Chip Mining. This process optimises the models ability to detect small objects by training the model specifically on areas where ground-truths are clustered together densely (chip areas). Patch-level augmentation is used to reduce the effect of class imbalance and the number of false positives. The developers use box voting in post-processing to further improve accuracy.

The most recent results for the VisDrone detection challenge 2021 [27] contains various other approaches to solving object detection in drone imagery. In this iteration of submissions, the highest performing algorithms were DBNet, SOLOer and Swin-T.

DBNet [27] is an adaptation of the Cascade R-CNN architecture which utilises Deformative Convolutions [28] in the 2nd, 3rd and 4th layers of the ResNet-101 backbone with a bottleneck ratio of 4. The developers also performs data augmentation in the way of random flips, shifting scale rotations , multi-scaling and centre cropping and image pre-processing through adding gaussian noise and randomly adjusting brightness and contrast to create more samples to train on.

The developers of SOLOer [27] use a scaled-YOLOv4 [29] backbone combined with a transformer based on the self-attention mechanism. To improve the models ability to detect small/low-resolution objects, they insert a two-way feature fusion network based on FPN, called BiFPN, into the models backbone.

The developers of Swin-T propose using a Swin Transformer [30] with a hierarchical design including a sliding window operation. The sliding window operation consists of two parts; a non-overlapping local window and an overlapping cross-window. This helps to combat the issue of scale variation in datasets and the high computational complexity when applying Transformers to high resolution images.

A running theme amongst the submissions to the VisDrone object detection competition is that the most successful algorithms (when looking at mean Average Precision and Average Recall metrics) make an effort to address the issue of small/low-resolution objects and high object density, be it through removing layers from the feature pyramid such as in the HAL-Retina-Net algorithm or the use of Augmented Chip Mining in ACM-OD.

The algorithm that currently holds the highest mAP score (53.5%) on the UAVDT benchmark [11] is FFAVOD Spot-Net with U-Net [31]. This model works by taking a collection of subsequent frames as input to a detection models backbone and then fusing the feature maps for each frame. These feature maps can then be used as standard by various detection architectures such as RetinaNet [3][11], however they propose improvements to the SpotNet architecture [32] and combine this with their feature fusion technique to obtain better detection results on [11].

2.4 Object Tracking

There are various different approaches to multiple object tracking, which generally take object detection, such as those outputted from YOLOv3, as input and then assigning each detection an ID which it attempts to keep the same across each frame if the detection belongs to the same object. The individual objects being racked across frames are often referred to as tracklets.

The SORT [33] algorithm tracks the detected objects by modelling their current states as vectors and using a Linear Velocity Model to estimate where the object will be in the next frame. When a detection is associated to a specific tracklet (i.e. a previously detected object), the location of the detected bounding box is used to optimally solve the tracklets velocity using a Kalman Filter [34] framework. To decide whether a detection should be associated with a tracklet, the tracklets new bounding box is estimated based on the velocity of the tracklet. The Intersection-over-Union (IOU) of the predicted bounding box and all detected bounding boxes are computed and the Hungarian Algorithm [35] is used to assign the predicted bounding box to a tracklet.

DeepSORT [36] is considered to be the state-of-the-art in multiple object tracking and extends upon the original SORT algorithm by incorporating the use of a deep neural Re-Identification (ReID) classification network to describe the appearence of objects such that they can be compared across frames. Incorporating this network reduces the number of identity switches between frames compared to in the base algorithm.

Another object tracking method that has recently gained popularity is the BYTETrack method [37]. BYTETrack deviates from most common MOT approaches as it utilises all of the proposed bounding boxes, whereas the likes of SORT and DeepSORT only look at boxes above a given threshold, otherwise ignoring them. The method itself works by first predicting the new locations of the tracklet using a Kalman Filter. The Intersection-over-Union of the detections bounding box and that of the predicted box is computed to find the motion similarity between them. This motion similarity is then used to match the detections with high confidence scores to tracklets. In the second phase, the remaining detections with low confidence scores are then matched to tracklets.

3 Methodology

The models and algorithms used for the experiments carried out in this project were created using the MMDetection [38] and MMTracking [39] libraries released by MMOpenLab. MMDetection and MMTracking are SDKs built on top of PyTorch that provide utilities that aid in the building, training, testing and analysis of object detection and tracking algorithms respectively. Models are built through configuration files with a modular design.

The majority of experiments were carried out on Durham University's NCC cluster. The cluster contains 134 CPU threads available across 5 CPU blades and 48 GPUs spread across 9 servers.

3.1 UAVDT

The experiments performed in this project will focus on the UAVDT dataset [11]. This dataset contains roughly $80k$ frames of footage captured using via aerial drones across 50 different scenarios and can be be used for Multi-Object Detection, Single-Object Tracking and Multi-Object Tracking. For our experiments we only consider the Multi-Object Detection and Multi-Object Tracking annotations in the dataset and we split the presented scenarios into a 60:40 train-test split. The dataset as a whole contains $840k$ ground-truths, representing 3 classes of vehicle; cars, buses and trucks (the vehicles class is only considered for MOD). The authors of the dataset also categorised each MOT scenario, via three attributes:

1) **Weather Conditions**
   Describes the weather conditions in which the footage was captured, split into three categories:
   - Day
   - Night
   - Fog
2) **Altitude**
   Describes the altitude of the drone when capturing the footage, split into three categories:
   - Low
   - Medium
   - High
3) **Camera View**
   Describes the position of the drone relative to the road on which the vehicles are driving, split into three categories:
   - Front-View (i.e. along the road)
   - Side-View (i.e. parallel to the road)
   - Bird-View (i.e. directly above the vehicles)

The annotations provide this information for both MOD and MOT with two additional categories for each individual object when looking at MOD. Firstly, the annotations provide information on whether the object is partially occluded, split into 4 categories:

- No occlusion (i.e. vehicle is fully visible)
- Small occlusion (i.e. 1% - 30% of the vehicle is blocked)
- Medium occlusion (i.e. 31% - 70% of the vehicle is blocked)
- Large occlusion (i.e. 71% - 100% of the vehicle is blocked)

Secondly, the information is provided on how much of the vehicle is out of bounds, split into three categories:

- Not Out-of-View (i.e. vehicle is fully in frame)
- Slightly Out-of-View (i.e. 1% - 30% of the vehicle is out of frame)
- Partially Out-of-View (i.e. 31% - 50% of the vehicle is blocked)

Any vehicles more that 50% out-of-view are ignored. Figure 4 from [11] shows the overall distribution of the UAVDT dataset as per each category previously mentioned.

We decided to use this dataset as it presented a variety of scenarios which covered all of the conditions in which one would expect drone-based traffic surveillance software to be deployed. The dataset does contain a large class imbalance, with cars representing the majority of the dataset. However, this too is representative of real-life conditions as cars are far more common on the road than trucks and buses, even though it does pose a challenge when training the detection models.

3.2 SOTA Object Detection

To fulfil our Basic Objective, we first train a state-of-the-art One-Stage detection algorithm and a state-of-the-art Two-Stage detection algorithm. The algorithms we decided to train were YOLOv3 [1] for One-Stage detection and Faster R-CNN for Two-Stage detection [4].

3.2.1 YOLOv3

The first algorithm, YOLOv3, [1] was chosen as our first One-Stage algorithm as research has shown that it boasts inference speeds significantly faster than other state-of-the-art algorithms such as SSD [2] and RetinaNet [3], whilst maintaining similar or better accuracy, even when working at a larger resolution. This property is especially useful in real-time applications such as traffic surveillance, as the detector is used as input to a tracking algorithm, and therefore allows the footage to be processed at a higher frame rate. This means that when the footage is reviewed it is smoother to watch and when being processed in real-time, there is less of a delay between when the video frame is captured and when the object is recognised by the drone, allowing it to take automated action more quickly.

In [1], the author proposes three different input sizes to the YOLOv3 algorithm; 320×320, 418×418 and 608×608. We elected to use the largest input size (608×608) as the dataset contains lots of small objects. Using a lower resolution input could make these objects harder to detect as some of their features, used by the algorithm to detect them, may be harder to distinguish or even lost entirely due to the loss of data.

YOLOv3 uses a new backbone architecture created by the authors called DarkNet-53 [1] which acts as the algorithms feature extractor and expands upon the DarkNet-19 architecture used in YOLOv2 [40]. The network uses consecutive 1×1 and 3×3 convolutions within residual blocks. Three different sized feature vectors are extracted from the network at size 76×76, 38×38 and 19×19. Using multiple feature vectors of varying size improves the ability of the algorithm to detect objects at a range of scales. Each feature map is passed to a distinct detection Head.

The each Head works by treating its extracted feature map as a [76×76, 38×38, 19×19] grid, whereby each cell in that grid is responsible for detecting exactly one object. The head generates a 3 anchor boxes of varying sizes per grid cell, with the size of the box depending on the resolution of the feature map. The 76×76 feature map uses anchor boxes of resolutions [10×13, 16×30, 33×23], the 38×38 map uses anchor boxes of resolution [30×61, 62×45, 59×119] and the 19×19 feature map uses anchor boxes of resolution [116×90, 156×198, 373×326]. Each anchor generated for each cell attempts to predict a bounding box, if there is believed to be an object present, represented as a vector containing the following values; the confidence of there being an object present in the box, the x and y coordinates of the box

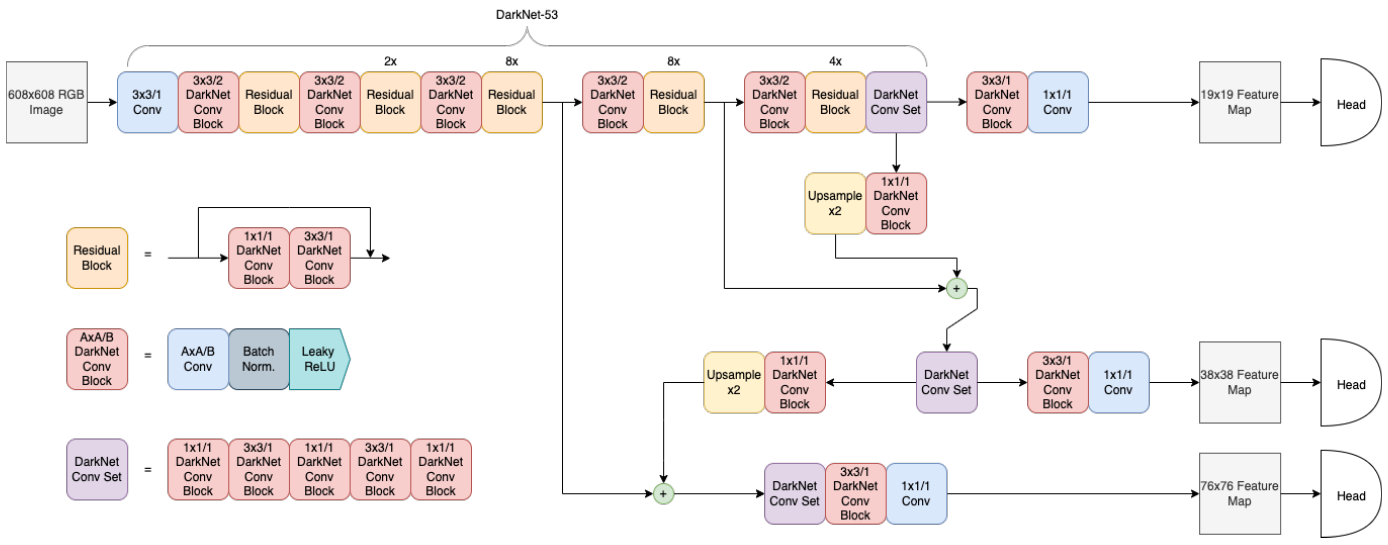

Fig. 3. Architecture of feature extraction in YOLOv3 on 608x608 image

relative to the cell center, the *height* and *width* values of the box relative to the dimensions of the anchor and a probability that it contains an object of each class. The head then uses regression to reduce the combined losses of the probability that an object is present, box dimensions and class probabilities. Predicted boxes undergo non-maximal suppression, which works by iteratively selecting the boxes with the highest confidence score and deleting all boxes which share a high IOU ($\geq$ 0.45) with the highest scoring box.

The model is trained to minimise a loss function that can be split into the same components as the vector which represents each bounding box. To calculate the loss of the x and y coordinates of the boxes centre, the class prediction and the confidence score (probability that there is an object in the box) use Cross-Entropy Loss. To calculate the loss of the regressed bounding box dimensions (*height* and *width*), Mean Squared Error is used. These 4 components are then summed to give a combined loss function.

When pre-processing the data going into the algorithm, we first resize the images to 608x608 so that they are compatible with the network architecture. Photo-metric distortion is then applied with randomised flips and the pixel values are normalised. the model was trained for 16 epochs and using Stochastic Gradient Descent with a learning rate of 1×10^{-3}. Pre-trained weights from a YOLOv3 model trained on the COCO dataset [41] (provided by mmdetection [38]) were loaded in to further boost the performance of the model.

3.2.2 Faster R-CNN

The second algorithm, Faster R-CNN [4], was chosen as our Two-Stage detector as it allowed for the use of Feature Pyramid Networks. The use of FPNs allow for the better better detection of smaller, lower-resolution objects. This is useful in drone imagery as many frames are captured at medium and higher altitudes and therefore objects appear smaller and are thus normally harder to detect.

The first stage of the model uses an FPN, containing a ResNet-101 backbone acting as the top-down side of the network, in order to produce 5 pyramidal feature layers $\{P_2, P_3, P_4, P_5, P_6\}$. The input image is passed through the ResNet-101 backbone and after the 9th, 21st, 90th and 99th residual blocks $\{C_2, C_3, C_4, C_5\}$, lateral connections are used to combine the outputs at this level with the bottom-up side of the network. For clarity, an overview of this process can be seen in Fig. 4. The outputted maps from $\{C_2, C_3, C_4, C_5\}$ contain $\{256, 512, 1024, 2048\}$ channels. These outputs undergo a 1x1 convolution, concatenation with the level above and then a final 3x3 convolution which results in each feature map $\{P_2, P_3, P_4, P_5\}$ having 256 channels. P_6 is obtained by sub-sampling P_5 with stride 2.

Each of the 5 feature maps are fed into their own RPN head, whereby each individual RPN head shares parameters with all others. The RPN heads contain a 256 channel feature layer, which is fully connected to both a bounding box regression layer and a classification layer, all of which need to be trained. The RPN head which is assigned to $\{P_2, P_3, P_4, P_5, P_6\}$ generates anchor boxes of resolution $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ which act as sliding windows across the feature map and are input into the RPN.

The second stage of Faster R-CNN [4] is the same as in Fast R-CNN [42]. At each level (except for P_6 which is ignored after the RPN stage), the proposed regions are passed through a distinct level-dependent Region-of-Interest Pooling layer, which uses a 7×7 scale and has an output dimension of 256. The output of this layer is passed through a fully-connected layer with an output dimension of 1024. From here, the outputted ROIs go through a fully-connected classification layer and bounding-box regression layer. The proposed bounding boxes then undergo non-maximum suppression in the same manner as in YOLOv3.

The first and second stages of this model use separate loss functions, however, there structure remains the same. When training the RPN and the output ROI head, L1Loss is used for bounding box regression, where as Cross-Entropy Loss is used for classification.

We train the model on the UAVDT dataset at multiple

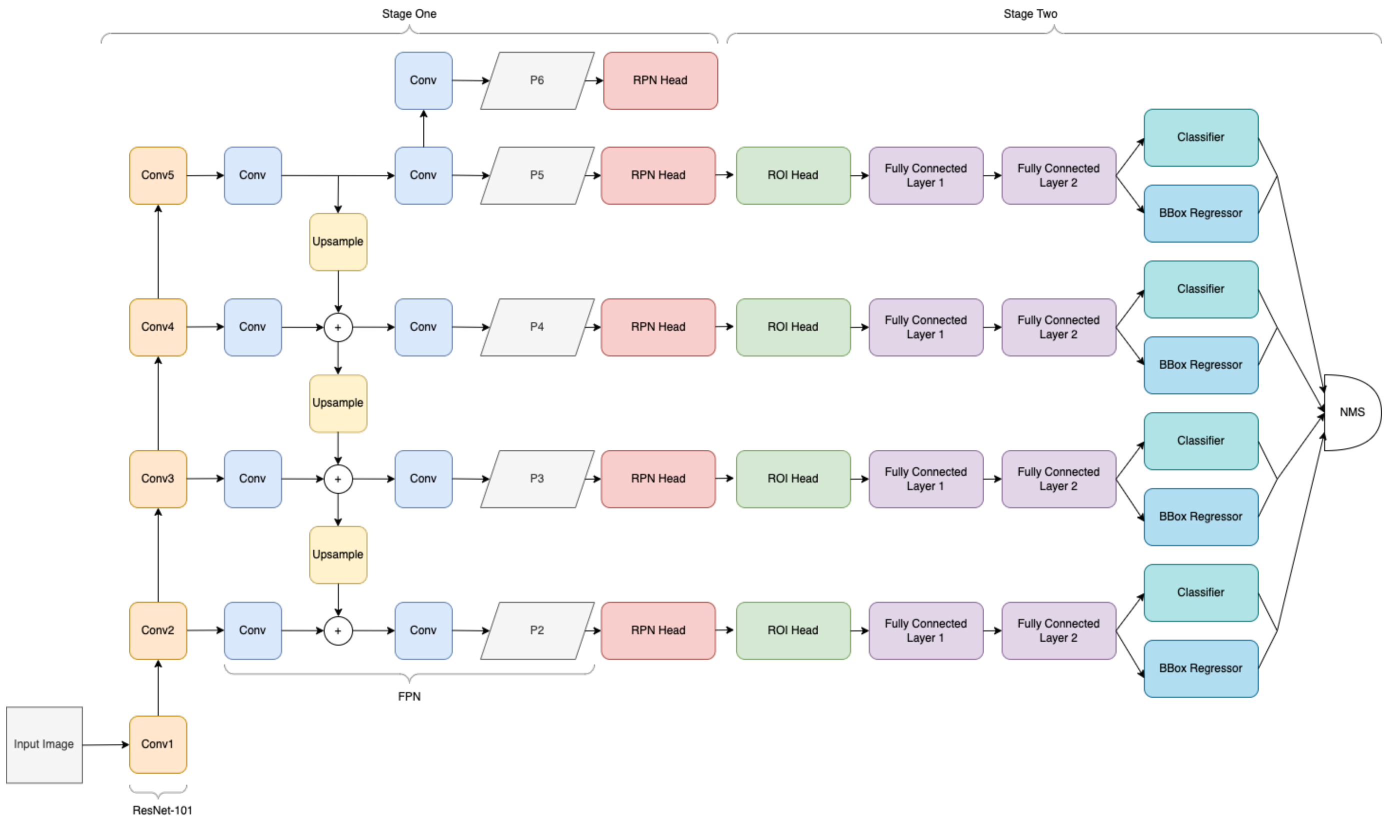

Fig. 4. Architecture of Faster R-CNN with ResNet backbone and FPN

scales; 1333×640, 1333×672, 1333×704, 1333×736, 1333×768 and 1333×800, the latter of which is also used at inference time. Random flips and normalisation are applied to the training. The model is trained for 12 epochs, uses a ResNet-101 backbone and takes advantage of weights transferred from a pre-trained Faster R-CNN model trained on the MS-COCO dataset [41] by mmdetection [38]). The model is optimised using Stochastic Gradient Descent with a learning rate of 1×10^{-2}.

3.3 Task Specific Object Detection

Whilst the current state-of-the-art algorithms we have trained so far boast good performance across most object detection benchmarks, the challenges presented by drone imagery regarding object detection, such as the common occurrence of low-resolution objects and high object density may cause them issues. In this section we look to train a one-stage detector and a two-stage detector which incorporate recent advancements to standard state-of-the-art algorithms such that they have characteristics which should help them overcome the challenges presented by drone-based imagery, as per this projects intermediate objective.

3.3.1 Free-Anchor RetinaNet

The first model we look to train that fulfils this criteria is an extension of the RetinaNet [3] model which incorporates the use of the Free-Anchor method [15], which we will refer to as FA-RetinaNet. Whilst the authors of [15] frame the method to be applied to any CNN architecture, the mmdetection framework [38] we use only allows for compatibility with RetinaNet, hence why we use RetinaNet to showcase the method as opposed to YOLOv3. Furthermore,the creators of the Free Anchor method [15], saw in their experimental results that their method, when applied to a RetinetNet detector, showed improved detection capabilities on slender objects and on images with a high object density, when compared with a vanilla RetinaNet implementation. Therefore, we expect to see that when using this model on UAVDT [11], the model should be able to make more accurate detections in areas with a high object density, such at a set of traffic lights where cars are closely compacted.

In order to understand the effect of incorporating the Free-Anchor method [15] into RetinaNet [3], it is important to first understand how the vanilla version of RetinaNet works. This model uses ResNet with FPN as a backbone to generate feature maps similar to the first stage of Faster R-CNN. However, instead of using pryamid levels $P2$ to $P6$, RetinaNet uses $P3$ to $P7$, where $P7$ is acquired by applying ReLU and an additional convolution to $P6$. Anchors are then assigned to each level whereby levels $\{P_3, P_4, P_5, P_6, P_7\}$ have anchors of size $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ respectively. These are then used at the aspect ratios 1:2, 1:1, 2:1 and additional anchors of $\{2^0, 2^{1/3}, 2^{2/3}\}$ the size of the anchors already applied at each level to further improve performance. This gives a total of 9 anchors per grid cell in each level. Each anchor is given a one-hot vector the size of the number of classes, representing the class within the detected box, as well as a vector of size 4 representing the bounding boxes x and y coordinates and $height$ and $width$ values.

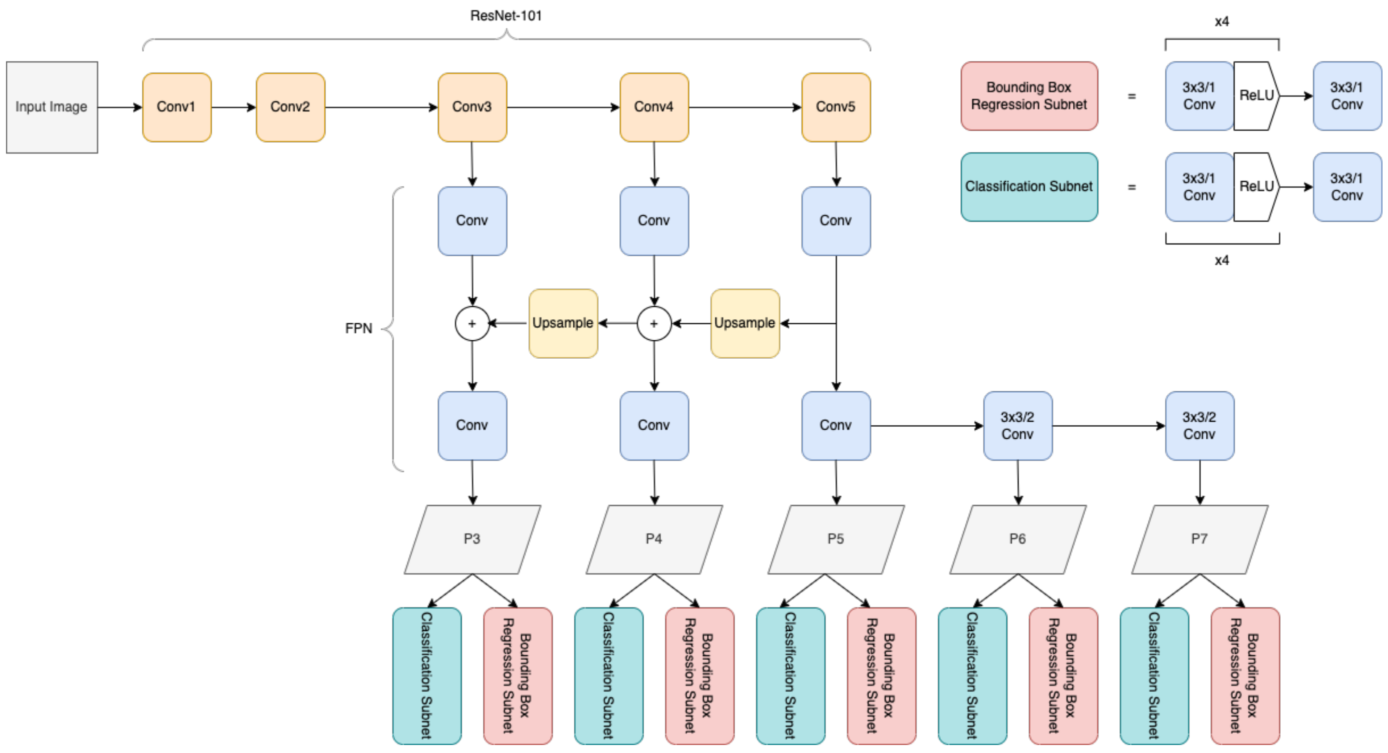

Fig. 5. Architecture of vanilla RetinaNet architecture with ResNet-101 backbone.

Instead of the using a region proposal network, each of the feature maps outputted by the FPN are fed straight into their own detection head. The heads contain a Classification Sub-net and a Bounding Box Regression Sub-net. Both networks contain 4 lots of 3×3 convolutions with ReLU activation functions followed by an additional 3×3 convolution. The two networks do, however, differ in their outputs, with the Regression Sub-net outputting the centre co-ordinates of the predicted box and its dimensions for each anchor, and the Classification Sub-net outputting the class predicted at each vector.

RetinaNet also incorporates an α-balanced version of the authors new loss function called Focal Loss which designed to reduce the effect of foreground-background class imbalance by down-weighting 'easy' detections and focusing on more difficult objects. This loss function replaces cross-entropy loss in the classification sub-net is defined as:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma log(p_t)$$

where p_t is the predicted probability that the object is of the ground-truth class, γ is the 'focusing' parameter which controls how heavily the 'harder' cases are weighted and α is a balancing factor for the ground-truth class. Standard L1Loss is used for the regression sub-net.

The Free Anchor [15] method alters the vanilla RetinaNet algorithm by changing the way that anchor boxes are matched. Instead of just assigning the anchor scoring the highest IOU to the object, first a bag of anchors is generated for each ground truth which contain the anchors with top-n IOU scores with the ground truth bounding box. Then the loss function is re-defined in order to allow the model to learn how to match anchors. This is achieved by defining a new loss function which incorporates the recall and precision of detections:

$$\begin{aligned} \mathrm{L}(\theta) = &-w_1 \textstyle\sum_i \log\left(\text{ Mean-max } (X_i)\right) \\ &+w_2 \sum_j FL\left(P\left\{a_j \in A_-\right\}\left(1 - \mathcal{P}_j^{bg}(\theta)\right)\right) \end{aligned}$$

where

$$X_i = \left\{\mathcal{P}_{ij}^{cls}(\theta)\mathcal{P}_{ij}^{loc}(\theta) \mid a_j \in A_i\right\} \quad (1)$$

[15] gives a full derivation of this loss function, however, in summary, it works as follows. The function is split into two components. The first component is the sum, for all ground truths, of the mean-max product of the classification confidence and localisation confidence of predicted bounding boxes generated from the anchors within that ground truths anchor bag. The second component is the sum, for each anchor, of the focal loss of the product of the probability that the anchor misses all ground truths and the confidence that the bounding box predicted by the anchor does not belong to the background. The second component then undergoes a weighted subtraction from the first to give the total loss.

We train the FA-RetinaNet model for 12 epochs using Stochastic Gradient descent and with a learning rate of 0.005 and a momentum of 0.9. A ResNet-101 backbone pre-trained on ImageNet [24] is used and pre-trained weights for the model itself are used, provided mmdetection [38]. The same multiple scale training is used as in the training of Faster R-CNN in the previous section. The original RetinaNet paper suggests a learning rate of 0.01, however our experiments found that this value would prevent the model from converging and so we decided to reduce it.

3.3.2 Faster R-CNN with CARAFE-based FPN

The second enhanced model we use is an extension of the Faster R-CNN model used as part of the basic objective. This model makes use of Content-Aware ReAssembly of FEatures (CARAFE) [17] which improves feature up-sampling. We choose this method as the creators found that when tested on the MS-COCO dataset [41], using CARAFE in place of nearest-neighbour interpolation when up-sampling in the top-down side of the FPN led to an increase in AP scores of small and medium objects by 1.8% and 1.4% respectively. As the UAVDT dataset comprises mostly of small and medium sized objects, we expect using this method should lead to improved detection results in our experiments.

CARAFE can be split into two components; the kernel prediction module and Context-Aware Re-assembly module. The kernel prediction module generates the kernels for each target location in the output feature map (i.e. the feature map being up-sampled to), which are used in the up-sampling process. This component can be further broken down into three sub-components; a channel compressor, a content encoder and a kernel normaliser.

The channel compressor reduces the number of channels per grid cell in the feature map to a predetermined hyperparameter value in order to minimise computational cost. The content encoder consists of a convolution layer into which the feature map is then fed in order to generate the reassembly kernels based on the content of input features. The generated kernels are then normalised by spatially using a soft-max function.

In the Context-Aware Re-assembly module, the generated kernels are then applied to their corresponding target location. Once the kernels have been generated, a weighted sum operator is used to apply each kernel to its corresponding target location and the square region around it in order to reassemble features.

3.4 Object Tracking

Having successfully trained detection models on the UAVDT dataset, completing the projects basic and intermediate objectives, we now look at applying these detectors to object tracking frameworks in order to complete the advanced objective. The models outlined in the previous sections are used to feed detections into the the tracking frameworks. We experimented with the DeepSORT [36] and BYTETrack [37] algorithms.

3.4.1 DeepSORT

The first object tracking framework we use is DeepSORT [36]. We chose this method as it uses deep CNN in-order to overcome the issue of instance re-identification. This is important in drone applications as the movement of the drone can result in perspective change which makes it more difficult to track the movement of objects when using methods that rely on velocity models and bounding-box overlap alone.

To understand how DeepSORT works, it is important to understand SORT [33], the algorithm on which DeepSORT advances upon. The SORT algorithm takes as input the detections from an object detection model such as the ones trained as part of the basic and intermediate objectives. A linear velocity model is then generated to approximate the target locations, box dimensions and velocity of the detected objects in the next frame. At the next time step, the predicted targets are associated with the new detections made at that step. The bounding box for that target is updated according to the new detection and a Kalman Filter framework [34] is used to update the targets velocity value. This values is used to optimise the linear velocity model.

Detected objects are then associated with targets by calculating the Intersection-Over-Union of their bounding boxes with the bounding boxes of the targets. The Hungarian algorithm [35] is then used to match the detected objects and targets based on the IOU metric.

The downside to this approach is that occlusions and sudden camera motion (like that encountered in drone footage due to the drones flying patterns) can result in the algorithm being unable to reconcile some objects with their corresponding targets due to poor IOU scores. These poor scores are obtained because the linear velocity model does not account for the issues such as camera motion and occlusion.

The DeepSORT algorithm [36], which we use in our experiments, improves upon this method by redefining the association mechanism used in SORT, replacing the IOU metric with a combination of two different metrics and using an alternative matching algorithm.

The first metric used is the Mahalanobis distance [43] between the detected object locations and the target locations. This alone, however, does not fix the outlined problems. Therefore, a second metric uses an appearance descriptor vector is used to how closely the detected objects 'look' to the target objects based on their appearance in previous frames.

This metric is achieved by taking the descriptor vector and comparing its cosine similarity with the set of descriptions for each target retained over the last 100 frames. The combined metric used to match the detected objects and the targets is the weighted sum of the Mahalanobis distance and the cosine distance between their appearance descriptors.

A gating value is calculated between each detection-target pair which is the product of two binary values, the first representing whether the Mahalanobis distance lies within a certain threshold value, and second representing whether the descriptor cosine distance lies within a certain threshold. If the product of the two values is 1, then the association is admissible. This is done to exclude unlikely associations.

The detection-target pairs are matched using a simple algorithm called matching cascade [36]. This works by sorting the set of targets into subsets by their age (i.e. how many frames they have been tracked across). The subset of oldest targets are taken first and the lowest cost matching is found with the unmatched detected objects. These matched object-target pairs accepted into a set of pairs if and only if the matching is admissible via the gating value being greater than 0.

The specific Re-Identification network architecture used in [36] is unavailable in the mmtracking library [39] so instead we train a ReID network with a ResNet-50 backbone pre-trained on ImageNet [24] to classify diffrent instances of 1049 distinct objects from the UAVDT dataset. To do this we converted the dataset into ReID format using utilities from [39]. The output of the ResNet-50 backbone is fed into a Linear ReID Head which takes the 2048 input channels and passes them through a 1024 channel fully-connected layer and finally through a 128 channel output. The network makes use of batch normalisation and ReLU activation. The model was trained for 6 epochs with the images scaled to 256x128. The model was optimised using stochastic gradient descent with a learning rate of 0.1, momentum of 0.9 and a weight decay of 0.0001.

3.4.2 *BYTETrack*

We also experimented with BYTETrack algorithm [37] as a second object tracking framework. This algorithm is significantly more simplistic than DeepSORT as it does not require any deep learning and so can process frames at a higher rate. This quality is particularly useful for the real-time applications of object detection on drone footage. Unlike most other object detection methods, the algorithm also attempts to use all provided bounding boxes instead of instantly discarding those with a low confidence score. This too is beneficial to the drone imagery domain as due to the dataset containing lots of low-resolution objects, the detections made by algorithms such as YOLOv3 and Faster R-CNN can have low confidence scores and so using BYTETrack means these objects will be tracked whereas when using some other tracking methods they may potentially be lost.

The algorithm itself works in multiple stages. Firstly, at each frame, the detections provided by used the detector are split into sets based on the confidence scores of the detection. In our experiments, detections with a confidence greater than 0.6 are put into the higher confidence set and detections with a confidence score between 0.1 and 0.6 are put into the lower confidence set. Anything with a confidence score below 0.1 is ignored.

Then, in a similar fashion to SORT, a Kalman Filter network is used to predict the locations of the targets from the previous frame. Detections from the set with high confidence are then associated with targets, using the Hungarian algorithm, based on the IOU of their bounding boxes. Detections and targets are only associated if their IOUs are greater than a threshold value (in our case 0.1). Any detections that have not been matched are moved into a new set of remaining detections and the same is done for any unmatched targets. The remaining targets are then matched with the detections from the set with low confidence scores again using the IOU metric and the Hungarian algorithm. In this case if the detections and targets have a IOU greater than 0.5 they are matched.

Any remaining targets are then assumed to be out of frame or occluded. They are retained for the next 30 frames in-case they re-appear (e.g. the vehicle comes out of the other side of a bridge) so that their identity is not lost. After the 30th frame the detection is removed from the set of active targets. Any of detections from the set of remaining detections (which had previously been in the set with high confidence scores) are then set up as new objects to be tracked in subsequent frames.

4 Results

4.1 Object Detection

4.1.1 *One-Stage vs Two-Stage*

Having trained all of the described object detection models, we then tested their performance on the test set of the UAVDT dataset. Various mAP metrics were calculated in order to obtain a good overview of the algorithms overall performance. We use the $mAP_{0.5}$ and $mAP_{0.75}$ metrics which quantify detections as true positives if they have an IOU with a ground truth bounding box greater than 50% and 75% respectively. We also use the $mAP_{0.5:0.95}$ metric which averages the mAP at IOU thresholds between 50% and 95% at 5% intervals. This is given for frames as a whole ($mAP_{0.5:0.95}$) and for 3 different scales of objects; small (mAP_s), medium (mAP_m) and large (mAP_l). We also tested their inference speeds, averaging the number of frames processed per second (FPS) across 2000 frames.

Rather unsurprisingly, we found that the two-stage algorithms out-performed the one-stage algorithms in terms of mAP but the one-stage algorithms boasted better inference speeds. When comparing the two models trained as part of the basic objectives (YOLOv3 and Faster R-CNN) we see that YOLO achieves an $mAP_{0.50}$ score of 27.8% whereas Faster R-CNN scores 28.8%. Given Faster R-CNN's use of a deeper ResNet-101 backbone and FPN in feature extraction as well as the use of RPN, better predictive performance was expected from the model. However, the YOLOv3 model boasted significantly better FPS rates, operating at 65.1fps as opposed to just 16.6fps in Faster R-CNN. This can also be attributed to the difference in architectural design, with the shallower DarkNet-53 backbone and only a single stage meaning that data can be processed much more quickly than in Faster R-CNN.

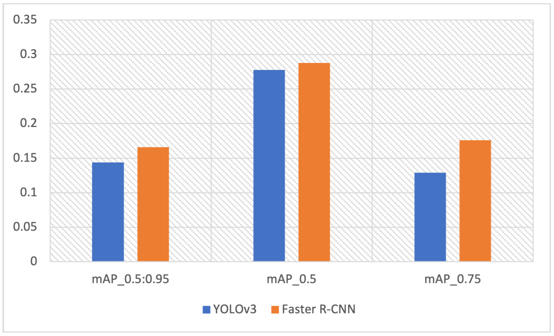

Fig. 6. Mean Average Precision scores of YOLOv3 and Faster R-CNN

4.1.2 *Enhanced Algorithms*

Our enhanced One-Stage algorithm, FA-RetinaNet, showed mixed results when compared to YOLOv3, as while YOLOv3 achieved a higher $mAP_{0.50}$ score of 27.8% compared to 25.9%, FA-RetinaNet achieved a higher $mAP_{0.75}$ score by 3.6%. This suggests that whilst YOLOv3 may

		$mAP_{0.5:0.95}$	$mAP_{0.5}$	$mAP_{0.75}$	mAP_s	mAP_m	mAP_l	FPS
One-Stage	YOLOv3	0.144	**0.278**	0.129	0.093	0.248	0.247	**65.1**
	FA-RetinaNet	**0.153**	0.259	**0.165**	**0.102**	**0.261**	**0.289**	16.6
Two-Stage	Faster R-CNN	0.166	0.288	0.176	0.105	0.283	0.261	**16.2**
	Faster R-CNN CARAFE	**0.175**	**0.306**	**0.185**	**0.115**	**0.297**	**0.271**	14.7

TABLE 1
mAP metrics for all object detection algorithms

provide more detections that can be considered valid, the FA-RetinaNet algorithm presents better quality detections (i.e. the regressed bounding boxes are closer to the ground truth).

Therefore, it can be argued that the FA-RetinaNet algorithm shows an improvement over YOLOv3 in this domain. Such an improvement was expected given that the creators of FA-RetinaNet claim improved performance by the algorithm in areas of high object density [15], which are frequently present in the UAVDT dataset. From a qualitative perspective, we can see evidence of this by visualising comparing sections of individual frames processed by each algorithm.

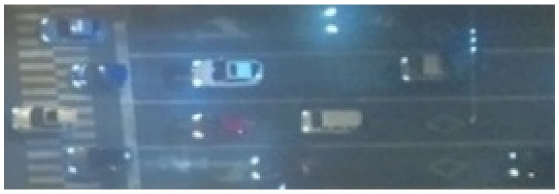

Fig. 7. YOLOv3 failing to detect any objects in this area of high density.

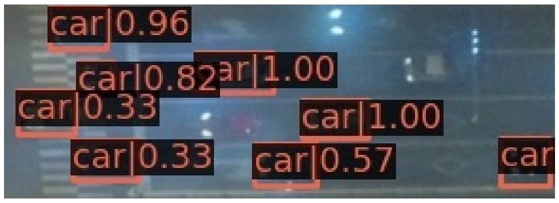

Fig. 8. FA-RetinaNet detecting 8 of the 11 present vehicles in the same area.

As Figures 7 and 8 show, the FA-RetinaNet model performs much better in this area of high object density, detecting 72% of the objects in the area, despite the poor lighting conditions, as opposed to YOLOv3 failing completely to detect anything.

The second enhanced model, Faster R-CNN with CARAFE based FPN, however, surpassed all the other models experimented with in terms of all mAP scores, achieving a score of 30.6% for $mAP_{0.50}$. We can also see an improvement in mAP across all object scales of at least 1%, the highest being 1.4% for medium scale objects, when compared to the Faster R-CNN model from the basic objectives. However, the improvement in mAP scores comes at the expense of a dip in fps by 9.3%. The reduction in fps can be attributed to the higher computational cost of the upsampling process used within the FPN architecture which utilises CARAFE instead of simpler nearest-neighbour interpolation.

4.1.3 Class Imbalance

While by the standard of common object detection challenges the results obtained by our experiments would be considered to show quite a low level of performance, these results actually fall in line with the experimental results found by the creators of the UAVDT dataset [11], where they found their most successful experiment, using R-FCN [44], only achieved a mAP of around 30% on the car class (the Faster R-CNN model we trained achieves 31.4% $mAP_{0.50:0.95}$ on the car class. The authors of [11] suggest that this may be caused by the high number of small objects caused from the higher altitude at which the footage is taken. Our results corroborate this suggestion as we can see that the mAP_s is very low for all algorithms (between 9-12%).

However, another reason for the low overall mAP score (which is averaged across all classes) could be the significant under-representation of the truck and bus instances in the dataset. Cars make up 96.1% of the object instances in the UAVDT dataset, as opposed to trucks and buses making up 2.0% and 1.9% respectively. Whilst UAVDT captures real world footage and so the data collected is representative of the real-life distribution of vehicles on the road the under-representation of these classes has likely adversely effected the models' predictive performance. To see the effects of under-representation we visualise the $mAP_{0.5}$ scores of the CARAFE based model for each class in the dataset.

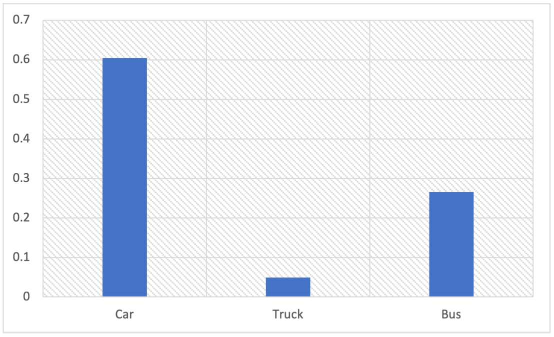

Fig. 9. Class-wise Average Precision of YOLOv3 and Faster R-CNN

As Fig. 9 shows, the model shows a very reasonable score for the car class, achieving an $AP_{0.5}$ score of 60.4%. In the bus category the algorithm also perform fairly well given that the class is under-represented in the dataset by a ratio of approximately 47:1, achieving an $AP_{0.5}$ of 26.6%. The truck class sees the worst performance with

	Car	Truck	Bus
YOLOv3	0.515	0.029	0.283
FA-RetinaNet	0.51	0.033	0.203
Faster R-CNN	0.577	0.053	0.219
CARAFE	0.604	0.049	0.266

TABLE 2
Class-wise $AP_{0.5}$ of all detection models.

	Car	Truck	Bus
YOLOv3	0.26	0.016	0.155
FA-RetinaNet	0.28	0.05	0.128
Faster R-CNN	0.314	0.037	0.147
CARAFE	0.322	0.029	0.173

TABLE 3
Class-wise $AP_{0.5:0.95}$ of all detection models.

the algorithm only achieving a $AP_{0.5}$ of 4.9%. Whilst the fact that both these classes have notably lower AP scores than the car class provides support for the fact that class imbalance has had a significant effect on the performance of the models, it is unlikely that this is the only factor due to the disparity between the two under-represented classes, which are under-represented by roughly the same degree.

On possible explanation for the disparity is the respective features of the two classes. It can be argued that, especially at low resolutions, trucks and cars share many similarities in their appearance, which may lead to the model incorrectly identifying trucks as cars. Due to the low-resolution of some of the vehicles, it's likely that lower-level features that make it easier to distinguish the two classes are not picked up by the feature extractors. Busses on the other hand look slightly more distinct than cars and trucks, with there more elongated shape and squared off corners, which should make it easier for the algorithms to distinguish.

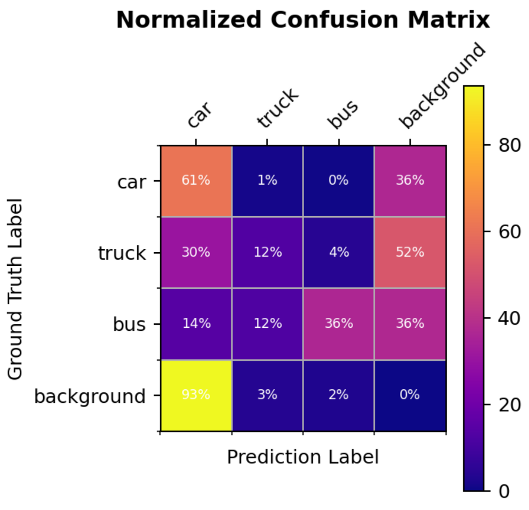

Fig. 10. Confusion matrix of classifications made by the CARAFE model

The confusion matrix shown in Fig. 10 provides support for these claims as it shows that whilst 30% of truck ground truths were misidentified as cars, compared to only 14% of busses being misidentified this way. The matrix also shows a significant portion of trucks and busses being identified as the background (i.e. missed detections) which could support the claims of class imbalance being a key factor.

4.2 Object Tracking

Within the mmtracking [39] environment, we also tested each of the object detection models as input to both DeepSORT [36] and BYTETrack [37] algorithms, obtaining a range of CLEAR MOT metrics [45]. We focus on the Mean Object Tracking Accuracy (MOTA) and Mean Object Tracking Precision (MOTP) metrics, the number of objects the algorithm was able to 'Mostly Track' (MT), 'Partially Track' (PT) and 'Mostly Lose' (ML), the number of ID swaps (IDs) on objects between frames and the number of False Positives (FP) and False Negatives (FN) given by the detector.

The MOTA metric is calculated from the number of False Negatives, False Positives, ID swaps and Ground Truths (GT) as follows:

$$MOTA = 1 - \frac{FP + FN + IDs}{GT}$$

The MOTP metric is calculated by finding the distance of the predicted bounding box from the ground truth for all objects across all frames. This distance is calculated as $1 - IOU$ of the two boxes. Table 4 shows a complete list of metrics for object tracking algorithm when paired with each of the detectors trained as part of the basic and intermediate objectives.

Our experiments have found that the BYTETrack algorithm out-performs DeepSORT for object tracking in the drone imagery domain, with the BYTETrack-CARAFE pairing showing the best overall performance. This combination of algorithms obtains the best MOTA score, the fewest False Negatives and the best results in terms of MT:PT:ML ratio. It also has the second best MOTP score, beaten by BYTETrack-YOLOv3 by just 0.1%.

In the general case, for each detection algorithm, when pairing BYTETrack proves to result in better performance across tracking metrics, with the BYTETrack pairings mostly achieving higher MOTA and MOTP scores, fewer ID swaps and better MT:PT:ML ratios than their DeepSORT counterparts, with the exception of the BYTETrack-FA-Retinanet pairing which achieves a slightly lower MOTA score.

A reduction in the number of ID swaps when using BYTETrack was to be expected given that the creators of the algorithm found such improvements over DeepSORT in their own paper [37], however the scale of the improvement is larger than expected.

As mentioned in section 3.4.2, this could be explained by the model not instantly discarding objects with a low confidence score, and instead attempting to use all available detections. As objects in drone footage are often occluded, blurred as a result of drone movement or small in scall due to higher flying altitudes, it is common for detections to have low confidence scores. DeepSORT discards detections if they have confidence less than 0.5 and therefore some correctly identified objects may be dismissed. As BYTETrack does not dismiss detections until its later stages, low-confidence objects can still be tracked resulting in fewer

		Rcll	Prcn	MT	PT	ML	FP	FN	IDs	MOTA	MOTP	FPS
DeepSORT	YOLOv3	0.468	**0.814**	452	**395**	411	**36644**	182531	2419	0.354	0.220	11.3
	Faster-RCNN	0.631	0.756	646	339	273	69752	126423	4700	0.414	0.220	7.6
	FreeAnchor	0.572	0.713	612	343	303	79046	146856	4873	0.327	0.214	10.9
	CARAFE	0.638	0.771	643	363	252	64896	124033	5213	0.434	0.221	7.4
BYTETrack	YOLOv3	0.504	0.809	523	340	395	40710	170022	**883**	0.383	0.223	**31.1**
	Faster-RCNN	0.64	0.747	651	334	273	74165	123324	954	0.421	0.221	18.4
	FreeAnchor	0.595	0.675	625	378	255	98259	138814	1674	0.304	0.218	18.9
	CARAFE	**0.65**	0.762	**654**	358	**246**	69618	**119974**	1076	**0.444**	**0.222**	17.4

TABLE 4
MOT metrics for each object tracking algorithm when paired with each object detection algorithm.

ID switches as well as accounting for the improvement in MOTA, MOTP and MT:PT:ML ratios.

5 Evaluation

5.1 Comparing Models

Having generated 4 potential solutions to the object detection via drone imagery problem and a further 8 for the object tracking problem, we now evaluate characteristics of the solutions and how well they would perform in practice. Regarding the task of object detection, we can see from our results that the two most impressive models are YOLOv3 and CARAFE-based Faster R-CNN.

Whilst the CARAFE-based model provides the best predictive performance, it does also have the lowest FPS rate of any of the models tested. YOLOv3 on the other hand has a significantly higher FPS rate, an increase of over 440%. In addition to this significantly better FPS rate, YOLOv3 also maintains a good level of predictive performance, differing from CARAFE by only 2.8% in terms of $mAP_{0.5}$.

Given the importance of FPS rate in the real-time applications of drone-based object detection/tracking, the precision-speed trade-off arguably falls in favour of the YOLOv3 algorithm. For example the DJI Air 2S Combo [46], an affordable camera drone that may be used for aerial photography, records at 60fps and so YOLOv3 is the only model that can process this footage in real-time. The other two algorithms, FA-RetinaNet and Faster R-CNN, whilst having a better FPS rates than the CARAFE-based model, are still very slow relative to YOLOv3, with FPS rates of 16.6 and 16.2 respectively and so also would not compete well with YOLOv3 in real-time tasks.

However, in applications in which footage is being reviewed after the event, for example watching back security footage, the use of a slower but better predictive model such as CARAFE can be justified as this application the footage does not need to be processed in real-time and better predictive performance would likely be preferential.

From an object tracking perspective, our results show BYTETrack to be the better performing algorithm in the drone footage domain. The reason for this is two-fold. Firstly, the algorithm has provides better tracking results over DeepSORT for most of the detection models we paired them with, partially due to BYTETrack using of all bounding boxes and not discarding objects with low confidence scores. Secondly, BYTETrack also boasts better FPS rates for all the detection models due to the lack of ReID network, meaning that the input frames undergo less processing.

As a result, in real-time applications, a solution of BYTETrack which takes detections from a YOLOv3 model as input would be the most appropriate, as this pairing has the highest frame-rate and maintains good object tracking performance, relative to the other models experimented with. However, similarly to with the object detection tasks, applications which would require processing the footage after it has been recorded (as opposed to on-the-fly) would benefit from pairing the BYTETrack algorithm with a CARAFE based-detection model, as this provides the best tracking performance and has a reasonable frame rate for non-real-time applications.

5.2 Reviewing the Detections

Reviewing the output of the various solutions on the test set of UAVDT, we can see, from a qualitative perspective, that there are certain scenarios where the solutions perform very well, and others where they struggle to perform. Due to constraints regarding the format of the paper, there is not enough room to showcase every single detection made by every single algorithm, however it is relevant to show some of the most notable results.

The best performance can be seen at low-medium altitude in clear day-time lighting conditions and is not noticeably effected by camera angle, i.e. front/side/birds-eye view.

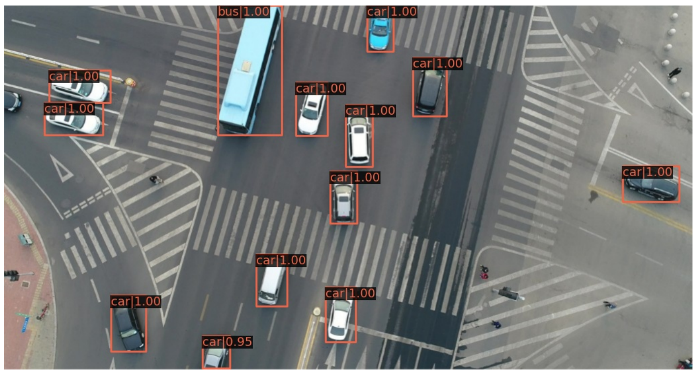

Fig. 11. Detections made by the vanilla Faster R-CNN model at medium altitude during the day.

Fig. 11 shows detections made by Faster R-CNN made under these conditions and depicts near perfect performance from the model. This level of performance was seen generally by all models in these conditions.

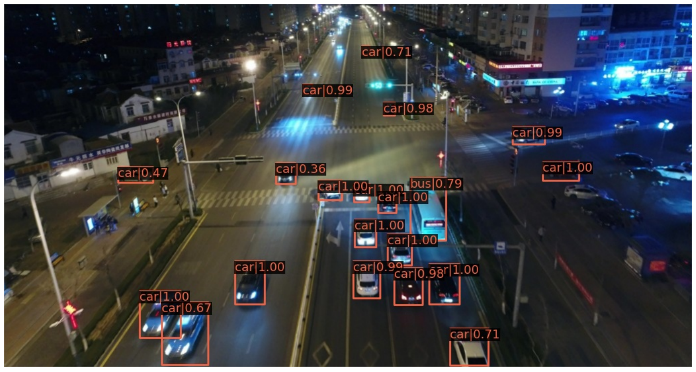
Fig. 12. Detections made by YOLOv3 at medium altitude at night.

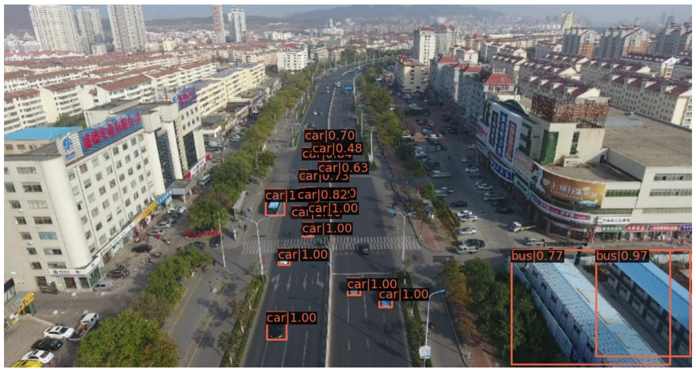
Fig. 14. Detections made by the FA-RetinaNet model which misidentifies long buildings as buses.

Good performance was also seen in night conditions at low to medium altitudes as depicted in Fig. 12, although some smaller vehicles weren't detected, perhaps as a result of glare from headlights.

One interesting scenario where we found all of the algorithms struggled to gain any kind decent performance is in the scenario depicted in Fig. 13, which shows some of the detections made by the CARAFE-based model.

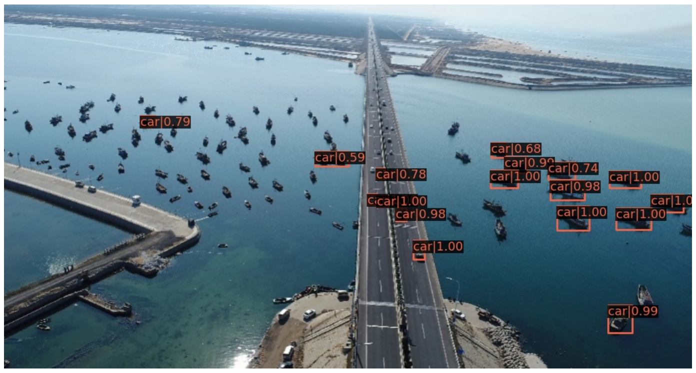
Fig. 13. Detections made by the CARAFE model in a scenario containing a bridge crossing a harbour.

Despite having the best predictive performance (metrically) of all the algorithms experimented with, even the CARAFE-based model still struggles with the frame in Fig. 13. There are two distinct failures that can be seen here. First of all, likely due to the very high altitude, one can see that the model struggles to detect some of the further-out vehicles on the road, despite them still being recognisable to the human eye. Another, more obvious failure, is that the model is incorrectly detecting boats/ships, which are not even instances in the dataset, as cars. Furthermore, in a lot of these cases, the boats are identified as cars with a high level of confidence. A potential explanation for this could be similar to that present for why many trucks were being misidentified as cars; at this high altitude the boats/ships may possess some spatial/temporal features that cars would possess at lower altitudes, thus confusing the detector as it is given no context regarding the circumstances (i.e. altitude, lighting etc.) in which the footage has been taken.

Another similar issue can be seen in Fig. 14, which shows the FA-RetinaNet model misidentifying a two story building as a bus. At first glance it is easy to see how this might have happened, as the buildings do resemble buses in their elongated shape and squared off edges. It is worth noting that the model actually performs well on the objects on the road despite the relatively high altitude, however it is likely that (similar to with the boats/ships) the buildings contain features respective of buses at lower altitudes.

5.3 Comparison to Other Results

The original UAVDT paper [11], carried out some of its own experiments using some other SOTA detection and tracking methods. However, the results they give are slightly confusing. First of all, the authors don't directly specify which mAP metric they are using, i.e. $mAP_{0.5}$, $mAP_{0.75}$ or $mAP_{0.5:0.95}$. Furthermore, they claim that the R-FCN implementation they use has an AP of 34.35%, however the figure that accompanies this claim (see Fig. 4 from [11]) actually suggests the mAP is lower, given that mAP is averaged across the 3 classes and both the truck and bus classes show AP scores of almost 0. Therefore, it is hard to directly evaluate the results for the experiments from this project against those from [11]. However, what is clear to see is that the SOTA models we have used do show obvious improvements in performance on the car and bus classes. Regardless of exactly which metric is being used in [11], the fact that all the models we use show $mAP_{0.5}$ and $mAP_{0.5:0.95}$ scores over 10% on the bus class for both metrics suggests improvement over their Faster R-CNN implementation, which differs to our model in that it uses a ResNet-50 backbone and has no FPN, and their implementations of R-FCN [44], RON [47] and SSD [2], all of which show scores of near 0.

Whilst our results do suggest better detection performance than the models used in [11], there are other algorithms which boast better predictive performance on the same dataset. FFAVOD-SpotNet with U-Net [31] boasts a $mAP_{0.5}$ of 53.5% and still holds the highest level of performance for this benchmark. Therefore, whilst the traditional SOTA detection algorithms do perform well on the drone imagery domain, there are some newer methods which exceed their performance, from a mAP standpoint. However, the authors of [31] don't provide any indication as to the models speed so therefore it is impossible to know whether the model is better for real-time applications.

From an object tracking perspective, it is much easier to draw comparison against the results in [11]. We see that the BYTETrack tracking algorithm when using the CARAFE-based Faster R-CNN as input outperforms all of their tracking implementations in terms of $MOTA$ and $\%MT$. However, they do present some implementations that offer better inference speeds and fewer ID swaps than this algorithm, albeit at a cost of tracking accuracy.

6 Conclusion

In this paper we have looked at how well SOTA one-stage and two-stage object detection algorithms apply to drone imagery and explored some recent advancements to SOTA object detection algorithms that can further improve their performance. We have found that whilst the two-stage detectors provide better predictive capabilities, the one-stage detectors operate at a higher frame rate whilst still maintaining a good level of predictive performance. Whilst the enhanced algorithms we suggest do provide a notable improvement in predictive performance over their basic one-stage and two-stage counterparts, we see that this comes at a cost in terms of frame rate. We have seen qualitative evidence that the FA-RetinaNet model is able to deal better with areas of high object density than the standard version of YOLOv3, the basic SOTA one-stage detector. We also see that replacing nearest-neighbour up-sampling in the FPN of Faster R-CNN's feature extractor with the CARAFE method improves its ability to detect small and medium scale objects, leading to better performance within the drone imagery domain.

Overall, we have seen that YOLOv3, whilst the weakest in terms of mAP scores, is still the most applicable detection algorithm in real-time drone imagery processing, out of the algorithms experimented with, as it operates approximately $4\times$ faster. Having said this, in applications where processing speed is less important, we found that the best solution from our experiments was the Faster R-CNN model which used a CARAFE-based FPN.

From an object tracking perspective we see that the BYTETrack algorithm out-performs DeepSORT in all aspects of tracking. We put this down to the fact that the BYTE-Track algorithm attempts to use all available detections, even those with a low confidence score which are normally ignored by DeepSORT. Furthermore, use of BYTETrack over DeepSORT is further justified by the fact it is able to operate at higher frame rates due to not requiring the use of a neural network, although there are implementations of BYTETrack that allow for neural network use that were not explored in this project.

Whilst this project has focused on vehicle tracking in particular, there are a range of potential applications for this research, not limited to the vehicle domain. Some such applications are discussed in the Introduction section such as; surveillance, agriculture, delivery and photography.

As we see an improvement in detection over the Faster R-CNN model used in [11] by using a ResNet-101 backbone with FPN, it would be interesting for future work to explore how altering the backbones used in feature extraction by various models can their effect performance within the drone imagery domain. One such experiment considered for this project as part of the intermediate objective was to see if the use of Swin Transformers [30] as backbones could improve the performance of the models used, given that they saw good performance in [27]. However, this was decided to be too far outside the brief of the project which aimed to focus on models built around CNN architectures of which Swin Transformers are not.

Overall, in answer to the original research question; *how well do state-of-the-art computer algorithms apply to the drone imagery domain and what methods can be used to gain improved performance?* we conclude that whilst improvements can be made to improve the performance of SOTA algorithms predictive performance on the drone imagery domain, they come at a cost of inference speed that leads to the traditional SOTA algorithms being more viable options in real-time applications.

Acknowledgements

This work has used Durham University's NCC cluster. NCC has been purchased through Durham University's strategic investment funds, and is installed and maintained by the Department of Computer Science.

References

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, pp. 21–37, Springer International Publishing, 2016.

[3] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jun 2017.

[5] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," 2017.

[6] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.

[7] I. Intelligence, "Drone market outlook in 2022: Industry growth trends and forecast," Apr 2022.

[8] K. J. Yaxley, K. F. Joiner, and H. Abbass, "Drone approach parameters leading to lower stress sheep flocking and movement: sky shepherding," *Scientific Reports*, vol. 11, p. 7803, Apr. 2021.

[9] "Increase in food delivery usage due to COVID-19 in great britain 2020, by age." https://www.statista.com/statistics/1107212/covid-19-food-delivery-frequency-in-great-britain/. Accessed: 2022-2-22.

[10] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu, "Vision meets drones: A challenge," *CoRR*, vol. abs/1804.07437, 2018.

[11] H. Yu, G. Li, W. Zhang, Q. Huang, D. Du, Q. Tian, and N. Sebe, "The unmanned aerial vehicle benchmark: Object detection, tracking and baseline," *International Journal of Computer Vision*, vol. 128, pp. 1141–1159, May 2020.

[12] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[14] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," 2016.

[15] X. Zhang, F. Wan, C. Liu, R. Ji, and Q. Ye, "FreeAnchor: Learning to match anchors for visual object detection," in *Neural Information Processing Systems*, 2019.

[16] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[17] J. Wang, K. Chen, R. Xu, Z. Liu, C. C. Loy, and D. Lin, "Carafe: Content-aware reassembly of features," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[18] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu, "Vision meets drones: A challenge," 2018.

[19] M. Mueller, N. Smith, and B. Ghanem, "A benchmark and simulator for uav tracking," vol. 9905, pp. 445–461, 10 2016.

[20] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Dota: A large-scale dataset for object detection in aerial images," 2017.

[21] P. Zhu, L. Wen, D. Du, X. Bian, H. Ling, Q. Hu, Q. Nie, H. Cheng, C. Liu, X. Liu, W. Ma, H. Wu, L. Wang, A. Schumann, C. Brown, C. Qian, C. Li, D. Li, E. Michail, F. Zhang, F. Ni, F. Zhu, G. Wang, H. Zhang, H. Deng, H. Liu, H. Wang, H. Qiu, H. Qi, H. Shi, H. Li, H. Xu, H. Lin, I. Kompatsiaris, J. Cheng, J. Wang, J. Yang, J. Zhou, J. Zhao, K. J. Joseph, K. Duan, K. Suresh, B. Ke, K. Wang, K. Avgerinakis, L. Sommer, L. Zhang, L. Yang, L. Cheng, L. Ma, L. Lu, L. Ding, M. Huang, N. K. Vedurupaka, N. Mamgain, N. Bansal, O. Acatay, P. Giannakeris, Q. Wang, Q. Zhao, Q. Huang, Q. Liu, Q. Cheng, Q. Sun, R. Laganière, S. Jiang, S. Wang, S. Wei, S. Wang, S. Vrochidis, S. Wang, T. Lee, U. Sajid, V. N. Balasubramanian, W. Li, W. Zhang, W. Wu, W. Ma, W. He, W. Yang, X. Chen, X. Sun, X. Luo, X. Lian, X. Li, Y. Kuai, Y. Li, Y. Luo, Y. Zhang, Y. Liu, Y. Li, Y. Wang, Y. Wang, Y. Wu, Y. Fan, Y. Wei, Y. Zhang, Z. Wang, Z. Wang, Z. Xia, Z. Cui, Z. He, Z. Deng, Z. Guo, and Z. Song, "Visdrone-det2018: The vision meets drone object detection in image challenge results," in *Computer Vision – ECCV 2018 Workshops* (L. Leal-Taixé and S. Roth, eds.), (Cham), pp. 437–468, Springer International Publishing, 2019.

[22] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms – improving object detection with one line of code," 2017.

[23] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[25] D. Du, P. Zhu, L. Wen, X. Bian, H. Lin, Q. Hu, T. Peng, J. Zheng, X. Wang, Y. Zhang, L. Bo, H. Shi, R. Zhu, A. Kumar, A. Li, A. Zinollayev, A. Askergaliyev, A. Schumann, B. Mao, B. Lee, C. Liu, C. Chen, C. Pan, C. Huo, D. Yu, D. Cong, D. Zeng, D. R. Pailla, D. Li, D. Wang, D. Cho, D. Zhang, F. Bai, G. Jose, G. Gao, G. Liu, H. Xiong, H. Qi, H. Wang, H. Qiu, H. Li, H. Lu, I. Kim, J. Kim, J. Shen, J. Lee, J. Ge, J. Xu, J. Zhou, J. Meier, J. W. Choi, J. Hu, J. Zhang, J. Huang, K. Huang, K. Wang, L. Sommer, L. Jin, L. Zhang, L. Huang, L. Sun, L. Steinmann, M. Jia, N. Xu, P. Zhang, Q. Chen, Q. Lv, Q. Liu, Q. Cheng, S. S. Chennamsetty, S. Chen, S. Wei, S. S. S. Kruthiventi, S. Hong, S. Kang, T. Wu, T. Feng, V. A. Kollerathu, W. Li, W. Dai, W. Qin, W. Wang, X. Wang, X. Chen, X. Chen, X. Sun, X. Zhang, X. Zhao, X. Zhang, X. Zhang, X. Chen, X. Wei, X. Zhang, Y. Li, Y. Chen, Y. H. Toh, Y. Zhang, Y. Zhu, Y. Zhong, Z. Wang, Z. Wang, Z. Song, and Z. Liu, "Visdrone-det2019: The vision meets drone object detection in image challenge results," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 213–226, 2019.

[26] J. Pang, K. Chen, J. Shi, H. Feng, W. Ouyang, and D. Lin, "Libra r-cnn: Towards balanced learning for object detection," 2019.

[27] Y. Cao, Z. He, L. Wang, W. Wang, Y. Yuan, D. Zhang, J. Zhang, P. Zhu, L. Van Gool, J. Han, S. Hoi, Q. Hu, M. Liu, C. Cheng, F. Liu, G. Cao, G. Li, H. Wang, J. He, J. Wan, Q. Wan, Q. Zhao, S. Lyu, W. Zhao, X. Lu, X. Zhu, Y. Liu, Y. Lv, Y. Ma, Y. Yang, Z. Wang, Z. Xu, Z. Luo, Z. Zhang, Z. Zhang, Z. Li, and Z. Zhang, "Visdrone-det2021: The vision meets drone object detection challenge results," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 2847–2854, 2021.

[28] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," 2017.

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[30] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021.

[31] H. Perreault, G.-A. Bilodeau, N. Saunier, and M. Héritier, "Ffavod: Feature fusion architecture for video object detection," 2021.

[32] H. Perreault, G.-A. Bilodeau, N. Saunier, and M. Héritier, "Spotnet: Self-attention multi-task network for object detection," 2020.

[33] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, IEEE, 2016.

[34]

[35] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, March 1955.

[36] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*, pp. 3645–3649, IEEE, 2017.

[37] Y. Zhang, P. Sun, Y. Jiang, D. Yu, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," 2021.

[38] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[39] M. Contributors, "MMTracking: OpenMMLab video perception toolbox and benchmark." https://github.com/open-mmlab/mmtracking, 2020.

[40] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016.

[41] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014.

[42] R. Girshick, "Fast r-cnn," 2015.

[43] G. Mclachlan, "Mahalanobis distance," *Resonance*, vol. 4, pp. 20–26, 06 1999.

[44] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," 2016.

[45] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, 01 2008.

[46] "Buy dji air 2s combo - grey: Drones."

[47] T. Kong, F. Sun, A. Yao, H. Liu, M. Lu, and Y. Chen, "Ron: Reverse connection with objectness prior networks for object detection," 2017.