pp 1871–1896.
doi:10.1017/aer.2021.43

Improving real-time drone detection for counter-drone systems

E. Çetin
ender.cetin@upc.edu
Technical University of Catalonia
UPC BarcelonaTech
Computer Architecture Department
Barcelona
Spain

C. Barrado and E. Pastor
Technical University of Catalonia
UPC BarcelonaTech
Computer Architecture Department
Barcelona
Spain

ABSTRACT

The number of unmanned aerial vehicles (UAVs, also known as drones) has increased dramatically in the airspace worldwide for tasks such as surveillance, reconnaissance, shipping and delivery. However, a small number of them, acting maliciously, can raise many security risks. Recent Artificial Intelligence (AI) capabilities for object detection can be very useful for the identification and classification of drones flying in the airspace and, in particular, are a good solution against malicious drones. A number of counter-drone solutions are being developed, but the cost of drone detection ground systems can also be very high, depending on the number of sensors deployed and powerful fusion algorithms. We propose a low-cost counter-drone solution composed uniquely by a guard-drone that should be able to detect, locate and eliminate any malicious drone. In this paper, a state-of-the-art object detection algorithm is used to train the system to detect drones. Three existing object detection models are improved by transfer learning and tested for real-time drone detection. Training is done with a new dataset of drone images, constructed automatically from a very realistic flight simulator. While flying, the guard-drone captures random images of the area, while at the same time, a malicious

Received 15 February 2021; revised 7 April 2021; accepted 27 April 2021.

drone is flying too. The drone images are auto-labelled using the location and attitude information available in the simulator for both drones. The world coordinates for the malicious drone position must then be projected into image pixel coordinates. The training and test results show a minimum accuracy improvement of 22% with respect to state-of-the-art object detection models, representing promising results that enable a step towards the construction of a fully autonomous counter-drone system.

Keywords: Counter-Drone; UAV; Drones; Object Detection; YOLO; EfficientNet; deep learning; Airsim

NOMENCLATURE

UAV	Unmanned Aerial Vehicle
AI	Artificial Intelligence
DRL	Deep Reinforcement Learning
RL	Reinforcement Learning
TL	Transfer Learning
CNN	Convolutional Neural Network
BiFPN	Bi-directional Feature Pyramid Network
API	Application Programming Interface
YOLO	You Only Look Once
LIDAR	Light Detection and Ranging
RPN	Region Proposal Network
ROI	Region of Interest
SVM	Support Vector Machines
HOG	Histogram of Oriented Gradient
FLD	Fisher Linear Discriminant
RGN	Relational Graph Network
GPS	Global Positioning System
GPU	Graphical Processing Unit
TPU	Tensor Processing Unit
φ	*Roll Angle in radians*
θ	*Pitch Angle in radians*
Ψ	*Yaw Angle in radians*
mAP	Mean Average Precision
IoU	Intersection Over Union
TP	True Positives
FP	False Positives
TN	True Negatives
FN	False Negatives
ms	milliseconds
BFLOPS	Billions of Floating-Point Operations required per Second

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

1.0 INTRODUCTION

Drones are generally used for surveillance, reconnaissance, shipping and delivery. Also, the number of drones which are commercially available is increasing, along with the risk of their misuse. Counter-drone systems are an emerging need to detect and eliminate malicious drones or any kind of UAV that threaten public security or individual privacy. Technologies for the detection, localisation and identification of small UAVs include infrared sensors, laser devices, optical surveillance aids and devices, acoustic devices, Light Detection and Ranging (LiDAR) sensors, equipment operating with image recognition technology, devices capable of detecting and localising UAV remote control signals and human air observers[1]. After a target drone is detected, elimination methods such as laser guns, water cannons, birds trained to catch drones and jamming can be applied.

However, drones themselves can also be used to counter malicious drones. In the literature, researchers have studied different instruments to detect UAVs. Choi et al.[2] proposed a radar system to detect drones such as quadcopters from long distances. The drone detection system was also tested experimentally in outdoor environments to verify its ability for long-range drone detection. In other research by Bernardini et al.[3], an acoustic drone detection method was presented. A machine-learning-based warning system was developed to detect drones by using their audio fingerprint. The effectiveness of this sensing approach was supported by preliminary experimental results. Haag et al.[4] presented LiDAR and radar sensors to detect small unmanned aerial system platforms. The position and average velocity of the target could also be determined very accurately by applying motion compensation and target tracking techniques based on the high update rate and ranging accuracy of LiDARs. A full counter-drone system using several types of sensors and several levels of prediction and fusion was also presented by Samaras et al.[5]. A Deep Reinforcement Learning (DRL) solution was proposed by Çetin et al.[6] to counter a drone by using another drone. The countering drone can autonomously avoid all kinds of obstacles (trees, cars, houses, etc.) inside a suburban neighbourhood environment, while trying to catch a malicious drone that is moving randomly. The current research could be considered as part of the object detection system in that DRL solution.

In this paper, a state-of-the-art object detection algorithm is trained and tested for the detection of drones in real time. Moreover, the drone detection models are trained using different kinds of image sets, one of which was created by capturing images from the AirSim simulator and auto-labelled. The results are analysed and compared against those of existing drone detection models described in literature. The proposed models are adapted to detect new types of drones by improving the layer parameters of a pre-trained Convolutional Neural Network (CNN). The remainder of this manuscript is organised as follows. In Section 2, related work is explained. Section 3 explains the training setup, drone image dataset and auto-labelling process, the real-time object detection method and the models used to detect drones and transfer learning, as well as the mapping from world to image coordinates. In Section 4, the training and test results are presented. Also, the performance of the proposed models is explained and analysed in detail, followed by a discussion and the conclusions of this work.

2.0 RELATED WORK

In this section, firstly studies related to state-of-the-art object detection methods are presented, then drone detection models available in literature are explained.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

2.1 Object detection models

Object detection is one of the core techniques used in computer vision and image processing. There are several fast and powerful real-time object detection systems, such as Fast r-CNN[7], Faster r-CNN[8], Xception[9], Yolo[10], or EfficientNet[11][12]. These methods have been updated and represent considerable improvements with respect to former CNN models. For example, Fast r-CNN[7] is an evolution of the VGG16 network, a region-based CNN, using the Caffe framework with multitasking, in which training is carried out in a single stage, thus avoiding any disk storage. Faster r-CNN[8] builds on Fast r-CNN by introducing a Region Proposal Network (RPN) with the aim of proposing regions of interest at the same time that feature maps are being generated.

Xception[9] is also an evolution of the VGG16 network that uses inception layers[13]. These are neural network layers that independently look for correlations across channels and at spatial pixels. Xception is based on a linear pipeline of depth-wise separable convolution layers, efficiently implemented in TensorFlow and forms the base of the Facebook object detector software.

Yolo[10] is a family of algorithms used in many research studies on object detection. Proposed by Redmon et al., Yolo, named after the slogan "you look only once", frames object detection as a regression problem to spatially separate bounding boxes and associated class probabilities. Yolo processes images by first resizing the input image, then secondly, a single convolutional neural network runs on the image. Finally, the resulting detections are thresholded based on the model's confidence. Newer versions of Yolo propose the use of larger neural networks than the precious version and/or show faster execution. Yolo predicts bounding boxes using dimensional clusters as anchor boxes[14]. Yolo offers the advantage of allowing multi-label predictions; this is, an object can be detected as two (or more) different labels at the same time. In this way, a friendly drone and a malicious drone can both be detected and labelled as drones, while at the same time, if their visual appearance is known, they could also be labelled as a threat or non-threat.

EfficientNet[11] is a brand-new state-of-the-art object detection model that, together with its recent evolution, EfficientNet[12], has become very popular in a short time thanks to its accuracy and efficiency. One of the key improvements is its novel Bi-directional Feature Pyramid Network (BiFPN), which allows information to flow in different directions: top down and bottom up. Secondly, EfficientNet uses a fast, normalised fusion technique, which adds an additional weight for each input feature, thus identifying the importance of each input feature. Finally, it introduces a scaling method, which jointly resizes the resolution/depth/width of the model to better fit with different resource constraints. EfficientNet-B0 is a version of EfficientNet adapted for small-size objects.

2.2 Object detection models for on-board UAV processing

Object detection methods have also been implemented to detect objects from UAVs with different target applications such as surveillance and disaster management.

For example, real-time object detection has been performed to detect humans by using videos captured from a UAV[15], addressing constraints such as computation time, viewing scale and altitude. The results are also visualised in a Geographic Information System platform by geo-localising objects to world coordinates.

In other research, a technique was proposed[16] to detect humans at a high frame rate by using an autonomous UAV. A map of points of interest is built by geo-locating the positions

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

of the detected humans. The video sequences, which are streamed from two video sources, thermal and colour, are collected on board the UAV and fused to increase the successful detection rate.

Furthermore, Yang et al.[17] recently proposed a real-time detection and warning system based on artificial intelligence (AI) to monitor social distancing between people during the coronavirus disease 2019 (COVID-19) pandamic[18]. A fixed monocular camera is used to detect individuals in a region of interest (ROI), and the distances between them are measured in real time without data recording. The proposed method was tested across real-world datasets to measure its generality and performance.

2.3 Drone detection models

In 2017, the European SafeShore project, in parallel with the IEEE AVSS conference, launched the *Drone-vs-Bird Challenge* to improve methods for detecting UAVs close to coastal borders, where they can easily be confused with birds. The challenge aims to correctly label drones and birds appearing in a video stream. From the papers presented in the first and second editions, in 2017 and 2019, respectively, it can be concluded that new advances are being driven by using CNNs with more and more layers, larger training datasets and improved implementations. In addition, the exploitation of temporal information is a key issue in differentiating drones from birds. The best paper of 2019[19] proposed a 110-layer CNN based on a semantic segmentation U-Net network originally designed for medical image processing, adding some dilation layers to improve detection of small objects. Posterior spatial–temporal filtering allowed an F1-score of 0.73 to be obtained. The set of training and test images used in those works, although challenging, were taken from the ground with the sky as background.

Drone-Net[20] is a deep learning model, available as open source, trained with 2,664 real drone images. It contains 24 convolutional layers in total, two of which are detection layers called Yolo layers. The details of the Drone-Net CNN are shown in Fig. B.1 in the Appendix. This model is considered as the state-of-the-art model for drone detection, and its accuracy results will be compared with those achieved by the new models presented herein.

Xiaoping et al.[21] proposed a dynamic drone detection method based on two consecutive inter-frame differences. They combine a Support Vector Machines (SVM) classifier with the traditional Histogram of Oriented Gradient (HOG) detection algorithm, and add an intermediate step based on a Fisher Linear Discriminant (FLD) to reduce the dimensionality of the HOG features. Using a dataset of 500 images, their results show an accuracy above 90%, similar to other drone detection algorithms, but with improvements in terms of the detection time that allow the processing of up to 10 images per second. However, since this method is based on the difference between consecutive images, it is not suitable for a moving camera.

Hu et al.[22] proposed an object detection method, called DiagonalNet, by using an improved hourglass CNN as its backbone network and generating confidence diagonal lines as the detection result. A large dataset (10,974 sample images) was created by processing videos and photos with different backgrounds and lighting, augmented by rotating and flipping each image with random angles. The images contain six types of UAVs, including multirotor and helicopter devices, and are labelled manually. The proposed algorithm detects UAV quickly (at 31 images per second) and accurately (with mean average precision above 90%). However, the experiments were all carried out indoors and in close proximity to the target drone.

With the objective of improving swarm cooperative flight and low-altitude security, Jin et al.[23] proposed a Six-Dimensional (6D) pose estimation algorithm for quadrotors.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

The proposed algorithm, based on the Xception network and pre-trained with ImageNet, includes a novel Relational Graph Network (RGN) to improve the performance of the drone pose detection. The pose is obtained by recognising eight key points (nose, rotors, etc.) of a quadrotor. After training with 340 images, simulation experiments showed a mean average precision of 0.94 in position and 0.74 in velocity, representing an enhancement of 9.7% compared with the baseline network. Given the real-time capabilities of the algorithm (30 frames per second), it was also tested with real flights. However, the mean average precision dropped to 0.65–0.75. Moreover, the accuracy of the 6D pose results decreased for small-sized drones.

Carrio et al.[24] used AirSim to train a CNN detection network with 16 layers by automatically labelling depth maps. Depth maps were obtained by stereo-matching of the Red–Green–Blue (RGB) image pairs of the virtual ZED stereo camera on the AirSim drone. The ground-truth labels of the depth maps were generated automatically by colour segmentation of the visual image. After training with 470 images, the detection system was integrated on board a small drone and tested while the drone navigated in the environment. The results showed that the system can simultaneously detect drones of different sizes and shapes with mean average precision of 0.65–0.75 and localise them with a maximum error of 10% of the truth distance when flying in linear motion encounters. The solution is limited to a maximum distance of 8m with relative speeds up to 2.3m/s.

Wyder et al.[25] presented a UAV platform to detect and counter a small UAV in an indoor environment where Global Positioning System (GPS) is not available. An image dataset is used to train a Tiny Yolo object detection algorithm. This algorithm, combined with a simple visual-servoing approach, is also validated in a physical platform. It successfully tracked and followed a target drone, even with an object detection accuracy limited to 77%.

3.0 CONTRIBUTIONS

The main contribution of the current work is that the drone detection models are constructed by transfer learning and training a state-of-the-art object detection algorithm. The drone detection models are trained using different kinds of images of drones to obtain a more robust drone detector. The source codes are available online[26], and the model configuration is presented in detail. Images captured from the AirSim simulator are automatically labelled with a bounding box around the drone. One of the advantages is that the time to label each image captured in the simulator is reduced compared with labelling them manually. Besides, the images can be used directly for training without needing third-party applications for labelling. It is considered that the auto-labelling procedure introduced here can save time for many researchers who work in the object detection field. Finally, the models are also tested, and the overall test results show that the proposed models achieve acceptable accuracies above 85% to detect only drones.

4.0 TOOLS AND METHODS

In this section, the tools and methods that are used for developing, training and testing the neural network models are discussed.

4.1 Tools

The following training tools are used for model creation:

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

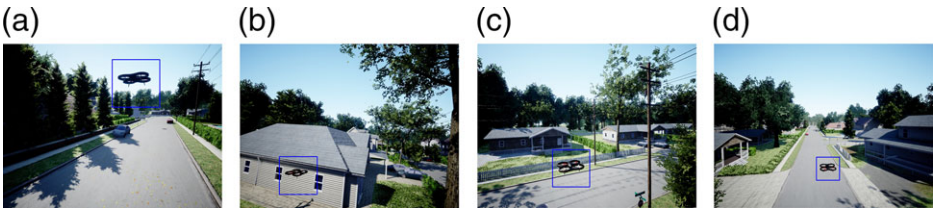

Figure 1. AirSim training images.

- AirSim simulator

AirSim[27] is a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles such as cars or drones. AirSim is built as an Unreal Engine[28] plugin. Unreal Engine provides ultra-realistic rendering and strong graphical features for AirSim. Many environments are available for use in AirSim. In this work, the suburban neighbourhood is selected to capture images.

- Darknet framework

Darknet[29,30] is a framework for training and testing of neural networks, written in C. The C language provides an efficient solution for general object detection in real time. We use Yolo-V3[31] with Darknet-53, the original 53-layer CNN shown in Fig. A.1, which achieves the highest measured floating point operations per second. In addition to the implementation of the algorithms, the Darknet framework also provides several pre-trained CNN models.

- Local desktop computer and Google Cloud Platform

The proposed models are trained on a desktop with an NVIDIA GeForce GTX 1060 graphical processing unit (GPU) with 6GB RAM graphic co-processor and Intel i7 processor with 16GB of memory. In addition, the Google Cloud Platform provides a colaboratory service[32] ("Google Colab"), which allows you to write and execute python in an internet browser. Google Colab allows the execution of codes on Google's cloud serves, which include powerful hardware including GPUs and tensor processing unit (TPUs). Neural networks were also tested on a Google cloud server with a Tesla T4-16GB GPU.

4.2 Drone image dataset and auto-labelling

Training a CNN requires a large dataset of labelled images. In this work, new drone images for the training dataset were captured by using the AirSim simulator. AirSim provides a public Application Programming Interface (API) for receiving parameters related to the drone and environment. We created a dataset of 2,000 images for training, $1,280 \times 960$ in size, captured in AirSim by using a drone flying randomly in the environment. Then, the captured images are auto-labelled by mapping the drones position in world coordinates to the image coordinates. Some of the image samples used in training can be seen in Fig. 1.

The procedure for projecting from 3D world coordinates to the image plane includes a few steps, which are explained in detail below:

- The pinhole camera model

The pinhole camera model defines the geometric relationship between a 3D point in the scene and its corresponding 2D projection onto the image plane. This geometric mapping

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

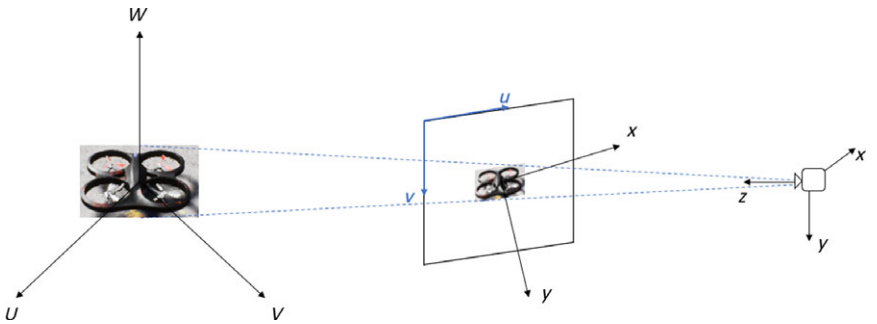

Figure 2. Pinhole camera projection visualisation.

Figure 3. Projection summary.

from 3D to 2D is a perspective projection. In the AirSim simulator, a pinhole camera model is available and mounted onto the drones for capturing images. The pinhole camera models in AirSim do not include the geometric distortion caused by lenses. Figure 2 shows a schematic view of the pinhole camera projection. The following paragraphs explain the different coordinate systems in more detail:

- Forward projection

The order of the forward projection is shown in Fig. 3. Firstly, the world coordinates of the drone which is found in the image are converted to camera coordinates by using quaternions and a rotation matrix. The rotations are described as a yaw–pitch–roll sequence, and the rotation matrix can be obtained by using the Euler angles which are available in the simulator as shown in Equation (1).

Quaternions are applied to the coordinate rotations and to relate them to the Euler angles[33]. The quaternion for a yaw–pitch–roll sequence are presented in Equation (2)

$$R = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\cos(\phi)\sin(\psi)+\sin(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\sin(\psi)+\sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi)+\cos(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\psi)+\cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix} \quad \ldots(1)$$

$$q = \begin{bmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \end{bmatrix} \quad \ldots(2)$$

The drone captures an image of the other drone, which is visible in the camera at certain angles when the field of view of the camera is less than 60°. Secondly, the image coordinates

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

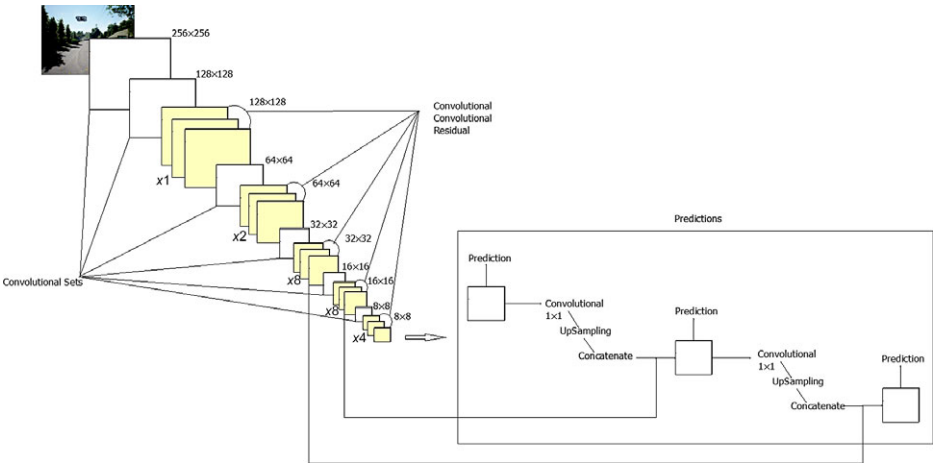

Figure 4. Darknet-53 CNN used as backbone for model 2.

are obtained by using the perspective projection of the camera, which uses the camera matrix received from the simulator. Finally, the pixel values are calculated by moving the origin to the upper-left corner of the screen. The mapping geometry is presented in Fig. 2.

4.3 Transfer learning

The main purpose of Transfer Learning (TL) is to improve the learning performance by using the experience from successfully pre-trained models[34]. TL can be used for different goals and in different situations. For instance, Drone-Net is built by using transfer learning to refine a Darknet model for detecting drones.

4.4 Proposed models

In this section, the new models proposed to improve the current results of Drone-Net are presented.

- Model 1 uses the same CNN architecture as Drone-Net, shown in Fig. B.1 in the Appendix. From the pre-trained Yolo network, up to 16 convolutional layers are transferred. After transferring these convolutional layers, the network is trained with 2,000 new images obtained from the AirSim simulator and another 1,000 images taken from the Drone-Net training set. The main purpose is to use the pre-trained weights from Drone-Net with the hope that the new model can detect real drones as well as drones from AirSim images.
- Model 2 is built by using the pre-trained weights from Darknet-53 for the neural network model based on the Yolo-v3[31] architecture. The neural network model is based on the Yolo-v3[31] architecture. In this model, the Darknet-53 model is implemented and the default Yolo-v3 network is modified to detect only the drone class. The Yolo-v3 network shown in Fig. 4 contains 107 layers: 75 convolutional layers, 23 shortcut layers, 4 routes, 2 upsamples and 3 Yolo detection layers. The predictions in Fig. 4 show that Yolo-v3 detects objects in three different layers. The model summary can be seen in Fig. C.1 of the Appendix. In this model, up to 74 convolutional layers from Darknet-53 model are transferred, then the training is extended with the 3,000 images: 2,000 images from the AirSim simulator as before, plus another 1,000 images taken from the Drone-Net training set.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 1
Backbone models used in training

Model	No. of CNN layers	No. of classes
Drone-Net[20]	24	1
Darknet-53[31]	107	80
EfficientNet-B0[11]	145	80

Table 2
Model details for training (all 6,000 iterations)

Model	Backbone (no. of pre-trained layers)	Training datasets	No. of images
Model 1	Drone-Net (16)	AirSim + Drone-Net	3,000
Model 2	Darknet-53 (74)	AirSim + Drone-Net	3,000
Model 3	EfficientNet-B0 (132)	AirSim + Drone-Net	3,000

- Model 3 is constructed and optimised by using the EfficientNet-B0 object detection algorithm to detect a drone. EfficientNet-B0 contains 145 layers and 2 detection layers. Up to 132 convolutional layers from Efficient-D0 model are transferred. The EfficientNet-B0 model summary used in training can be seen in Fig. 15 in the Appendix. The training set includes the 3,000 images: 2,000 images from the AirSim simulator and 1,000 images taken from the Drone-Net training set. The models proposed here have backbone networks such as Drone-Net, Darknet-53 and EfficientNet-B0. The details of these backbones are presented in Table 1. In addition, in Table 2, the model details are explained. For instance, model 1 uses up to 16 Drone-Net backbone network convolutional layers, and model 2 has a backbone network from Darknet-53 with up to 74 convolutional layers. Model 3 has more layers in total than the other models thanks to the EfficientNet-B0 model, which has 145 layers in total, and it uses up to 132 convolutional layers from EfficientNet-B0.

5.0 RESULTS

In this section, the models described in Section 4 are trained and tested. First, the training metrics are given, then the test results are presented.

5.1 Training results

Training was accomplished in 6,000 steps for all models. During training, the mean Average Precision (mAP) value is calculated and the best weights (those that give the highest mAP value) is saved. The mAP is the mean value of the average precision (AP) for each class, being the average precision, i.e., the area under the precision–recall curve[30]. The training results for all the models are summarised in Table 3, showing the mean average precision (mAP) metric and the training time for each model. The mAP is calculated for an Intersection Over Union (IoU) threshold of 0.5 and presented as mAP@0.5.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 3
Training results of models after 6,000 steps

Model	Best mAP@0.5%	Training time (h)
Model 1	83.63	2.5
Model 2	84.93	14
Model 3	89.77	18

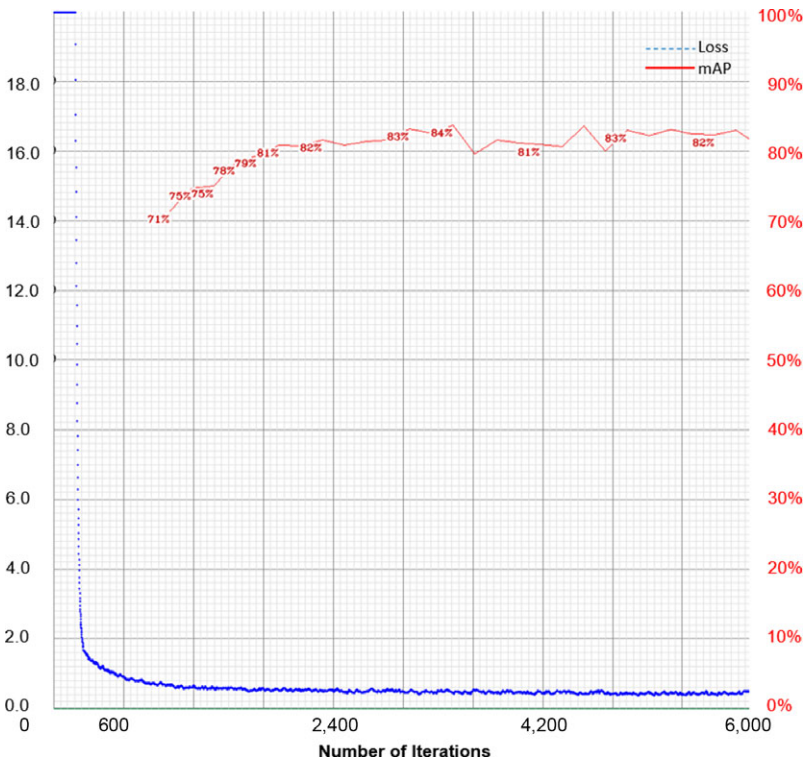

Figure 5. Loss and mAP (%) for training model 1.

These results show that model 3 achieves the highest mAP value of 89.77%, while model 1 reached the lowest mAP value of 83.63%. Model 2 has an intermediate mAP@0.5 value of 84.93% and also required a training time intermediate between model 1 and model 3. The training time can be affected by batch size and subdivisions, which is set in the configuration of each models for training. Model 1 had the shortest training time thanks to the size of its neural network, which has a total of 24 convolutional layers.

Training plots for each model are also presented in Figs 5, 6 and 7, where the red line represents the calculated mAP values and the blue line shows the loss value during training. Note that the loss value drops dramatically after 600 iterations for all the models. In Fig. 5, the model 1 mAP value starts at 71% and fluctuates around 80%. After 3,420 iterations, the best mAP value of 83.63% is recorded. In the meantime, the average loss value remains stable at the minimum of around 0.5, which is the highest among all the models.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

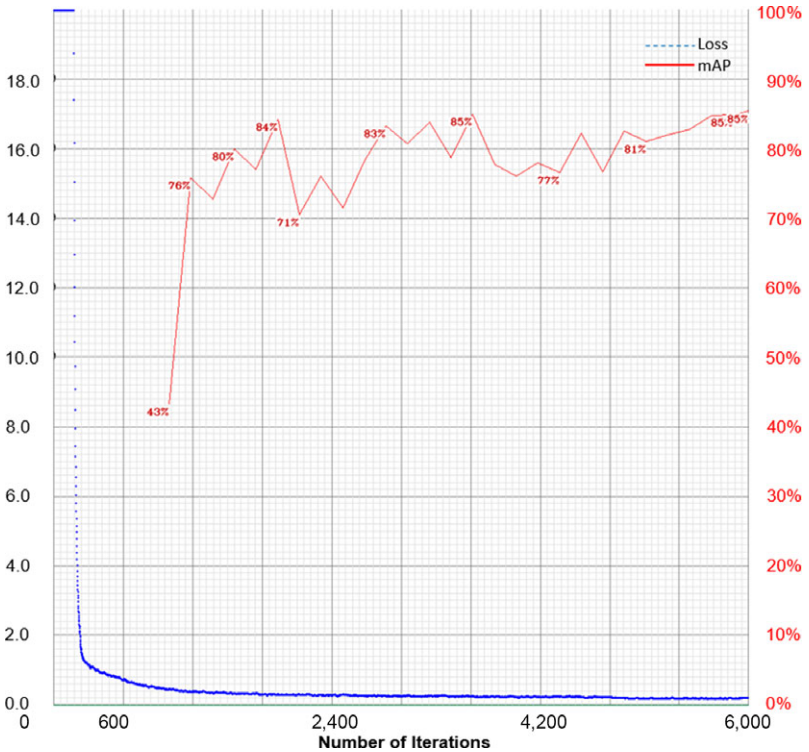

Figure 6. Loss and mAP (%) for training model 2.

Moreover, Fig. 6 shows the training progress for model 2. In this training, the mAP value starts at 43% and jumps to the 76% level. After 3,600 iterations, the best mAP@0.5 value of 84.93% is calculated. At the end of the training, the mAP value is calculated, but we saved the weights of the model for the first highest mAP value to avoid over-fitting.

Finally, the training progress for model 3 is presented in Fig. 7. As when training the other models, the mAP values were calculated during training and the best weights saved. In this training, the best weights were obtained at around 4,000 iterations. The model 3 mAP value reaches the highest value of 89.77% with respect to the two models described above. The mAP value of model 3 starts at a higher value of 68%, and the loss settles down to 0.25 in 6,000 iterations.

5.2 Test results

Once the models were trained, we fed them with new unseen images for the test evaluation. The state-of-the-art object detection models, shown as the backbone models in Table 1, and the three proposed models were tested with those different groups of images. The details regarding the number of test images from each group are presented in Table 4. Four sources of images were proposed with a balanced distribution of 20 images from each. Some of the images are shown in Fig. 8.

Note in Fig. 8 that the first two images (a, b) are from the AirSim simulator, images (c) and (d) are obtained from the Drone-Net test set while images (e) and (f) are random but challenging drone images, with noisy background, found in the Internet. Finally, the models

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 4
Example test images

Image source	No. of images
AirSim test images	20
Drone-Net test images	20
Web drone images	20
No-drone images	20

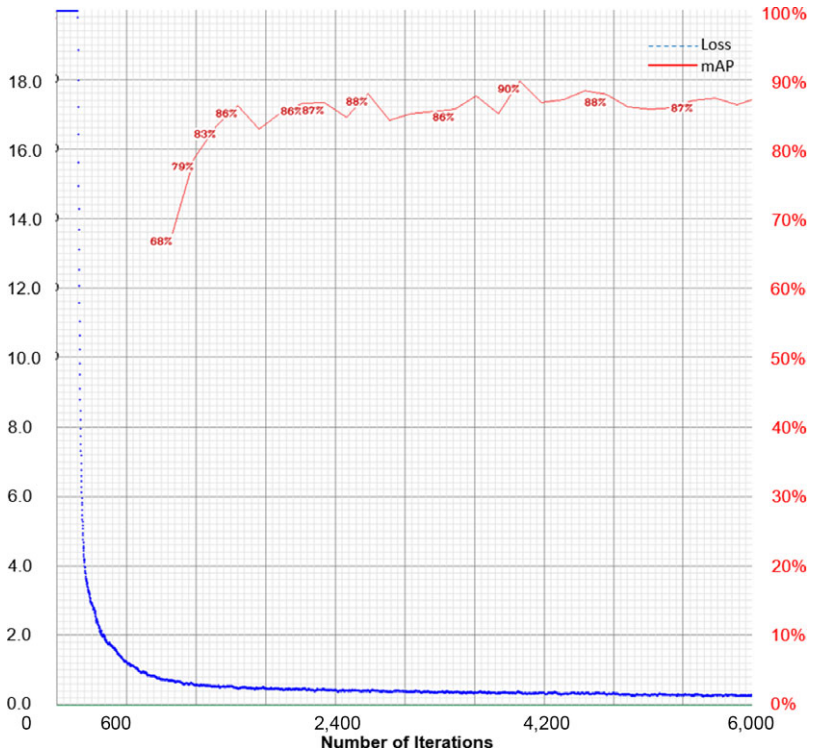

Figure 7. Loss and mAP (%) for training model 3.

were also tested with images, such as (g) and (h), which do not include any drones, to capture potential errors in prediction. Note that, unlike the other two sets, the AirSim images are drone images taken from another flying drone, not from the ground as in the other cases.

The following evaluation metrics were used to measure the test performance of the neural networks and compare the results obtained by each model:

- **Accuracy**: (TP + TN)/total predictions
- **Precision**: TP/(TP + FP)
- **Recall**: TP/(TP + FN)
- **F1-score**: 2 × (precision × recall)/(precision + recall)

based on the number of True Positives (TP), False Positives (FP) (or detection errors), False Negatives (FN) (or omissions) and True Negatives (TN).

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 5
Overall test results

Model	TP	TN	FP	FN	Accuracy (%)	F1-score
Drone-Net	37	18	7	18	69	0.75
Darknet-53	14	20	7	39	43	0.38
EfficientNet-B0	18	20	5	37	48	0.46
Model 1	52	16	6	6	85	0.90
Model 2	54	19	2	5	91	0.94
Model 3	53	16	5	6	86	0.91

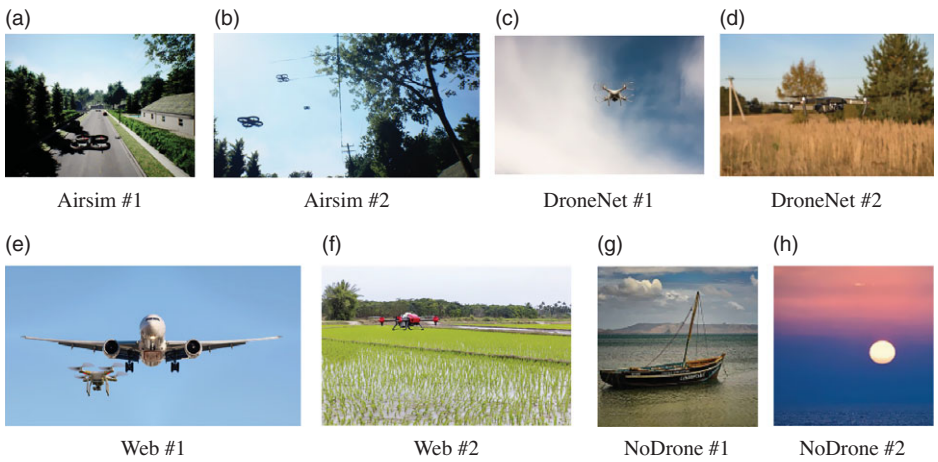

Figure 8. Test images from four different sets.

While the accuracy measures the goodness of the models, the precision and recall measure the errors in terms of the false detection rate and omissions, respectively. The F1-score, which is calculated as the harmonic mean of the precision and recall, is a measure of the robustness of a model.

The overall test results including the whole test set (80 images) are presented in Table 5. The test results for each set of test images are also analysed in detail, and the results presented in Tables 7, 8, 9 and 10.

The results presented in Table 5 show that Darknet-53 and EfficientNet-B0 were less accurate and, as expected, their F1-scores were very low compared with Drone-Net, which is a state-of-the-art drone detection model. The Darknet-53 and EfficientNet-B0 models were already trained by using the COCO[35] image set to detect up to 80 classes. These classes are different kinds of objects such as humans, cars, trees, birds, dogs, bags, trains, etc., while all kinds of air vehicle are labelled under a single class of aeroplanes. A detailed look at the number of TP predictions in Table 5 shows that Darknet-53 and EfficientNet-B0 miss most of the drone detections. For this reason, both models also have a high number of FN predictions and thus a low F1-score compared with Drone-Net. As a direct conclusion for counter-drone systems, it is not feasible to use such generic models directly to detect only drones. For this reason, the new models proposed here provide an improved way to detect only drones with acceptable accuracy. For example, model 1, model 2 and model 3 achieved high accuracies,

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 6
Comparison of real-time performance of the models

Model	Inference time (ms)	BFLOPS
Model 1	4.838	5.448
Model 2	40.756	65.304
Model 3	38.222	3.670

Table 7
AirSim image test results

Model	TP	TN	FP	FN	Accuracy (%)	F1-score
Model 1	14	0	1	5	70	0.82
Model 2	16	0	0	4	80	0.89
Model 3	14	0	0	6	70	0.83

Table 8
Web image test results

Model	TP	TN	FP	FN	Accuracy (%)	F1-score
Model 1	18	0	1	1	90	0.95
Model 2	18	0	1	1	90	0.95
Model 3	19	0	1	0	95	0.98

reaching 85%, 91% and 86%, respectively. Equivalently, the models have high F1-scores of 90%, 94% and 91%, respectively, with model 2 achieving the highest scores compared with the other models. This is very important because counter-drone systems are expected to detect drones precisely. In other words, the number of false detections is expected to be zero or as low as possible, thus avoiding failures in the detection of unexpected intruder drones. As seen in Table 5, model 2 achieved the lowest false detections compared with the other models.

The real-time performance of the proposed models is compared in Table 6. Evaluation metrics such as the inference time in milliseconds (ms) and Billions of Floating-Point Operations required per Second (BFLOPS) are used to compare each models. The models are tested on a 16-GB Tesla T4 GPU, which is commonly used among researchers. Model 1 achieved the fastest inference time of 4.838ms. Model 3 performed slightly faster than model 2, although model 3 has the highest number of layers, but the lowest BFLOPS value of 3.670.

To better understand how the models perform predictions, we carried out a deeper analysis of the results for each test dataset. The results of the AirSim images are presented separately in Table 7, revealing that the performance of the models was satisfactory for detecting AirSim images. Model 1 and model 3 showed accuracy of 70%, while model 2 achieved a higher rate of correct detection of 80%. Model 1 showed only one FP detection, while model 2 and model 3 showed none. The F1-score was also calculated for all the models. Model 2 showed the highest F1-score of 0.89, compared with 0.82 and 0.83 for model 1 and model 3, respectively.

Table 8 presents partial results for brand-new images taken from the Internet. Note that model 1, model 2 and model 3 showed very good rates of detection of 90%, 90% and 95%,

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

Table 9
Drone-Net image test results

Model	TP	TN	FP	FN	Accuracy (%)	F1-score
Model 1	20	0	0	0	100	1
Model 2	20	0	0	0	100	1
Model 3	20	0	0	0	100	1

Table 10
No-drone image test results

Model	TP	TN	FP	FN	Accuracy (%)	F1-score
Model 1	0	16	4	0	80	NA
Model 2	0	19	1	0	95	NA
Model 3	0	16	4	0	80	NA

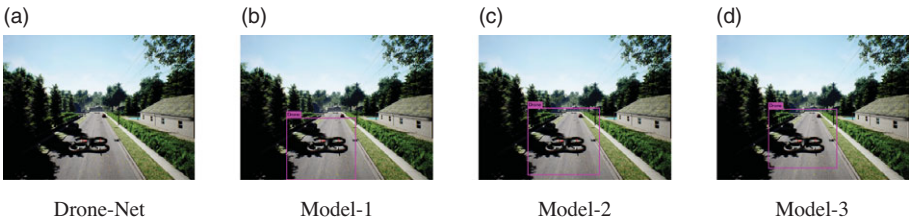

Figure 9. AirSim image test detection results.

respectively. Additionally, model 1, model 2 and model 3 achieved higher and promising F1-scores of 0.95, 0.95 and 0.98, respectively.

When looking at the partial results for the models tested with Drone-Net images, model 1, model 2 and model 3 detected the images with higher accuracy of 100%, as seen in Table 9. Also, the models showed higher F1-scores of 1.

Finally, the partial model test results for the images which do not include drones are presented in Table 10. It is expected that the models will not detect any object, including in images which include shapes similar to drones. All the models showed higher accuracy (above 80%), although model 1 and model 3 had few FP detections. The F1-score is undefined given that the number of TP is zero.

Figure 9 shows the same AirSim test image to compare the detection results of the four models able to predict only the class drone. The Drone-Net model prediction shown in Fig. 9(a) fails to detect drones in the AirSim test images. However, model 1, which has the same configuration as the Drone-Net model but is trained with AirSim images, successfully detected a drone (Fig. 9(b)). The model 2 and model 3 detection tests shown in Fig. 9(c) and (d) reveal that both correctly detected drones.

Figure 10 shows the test results of the models for a challenging image from the web image set. All the new models successfully detected the drone in the image. However, the state-of-the-art drone detection model Drone-Net failed to detect the drone against such a noisy background. In addition, the bounding box sizes can have different sizes. For example, in

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

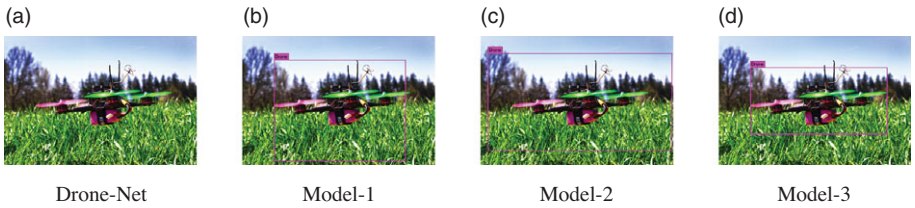

Figure 10. Web source image test detection results.

Figure 11. Inaccurate bounding boxes in auto-labelling.

Fig. 10(c), the bounding box is larger than expected, which can just cover the predicted drone. However, this is not the case in general in all the test images. Such different sizes of the bounding boxes may be the result of background noise and the scale of the drone dimensions.

6.0 DISCUSSION

In this section, further analysis is discussed.

6.1 Inaccurate bounding boxes of the auto-labelling process

In auto-labelling, it was observed that there were inaccurate bounding boxes, and they had to be removed from the training set. Almost 10% of the auto-labelled images were detected to be inaccurate. In Fig. 11, some of the inaccurately labelled images are presented. For instance, in Fig. 11(a), there is a bounding box at the top left of the image with no drone shown within. The position of the drone at the moment of the image capture was very close to the other drone, and the mapping from world to image coordinates was not correct. We believe that this issue was caused by the processing time between capturing the image and obtaining its world position from the simulator. The processing time does not cause problems when the drones are far from each other, but when they are too close, their relative speed is high and the retrieved drone location is already obsolete. However, if the drone stays stationary, this problem disappears and none of the bounding boxes are inaccurate. Other erroneous and disregarded bounding boxes, not centred correctly in the images, are shown in Fig. 11(b), (c) and (d), all of which correspond to a drone at the border or outside of the captured image.

6.2 FP detections

In this section, FP detections by the models are discussed and analysed. These are error cases where another object is mistakenly labelled as a drone. The FP images can be seen in Fig. E.1 in the Appendix.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

At the start of this research, we tested Drone-Net to detect drones in some images and found that most of them were FP, especially when dealing with AirSim images. Figure E.1(a), (b), (c), (d) and (e) shows AirSim test images detected as FP. These figures show that Drone-Net is not accurate enough to detect a drone in these images. The detections were bounded to objects such as trees or wires, or covering the whole image. In Fig. E.1, Drone-Net detects a drone, but it also detects the commercial aircraft as drone. Drone-Net also detects two of the images from the no-drone test image set as FP. Figure E.1(g) and (h) shows that objects such as humans and road signs are also detected as FP by Drone-Net.

Model 1 also detected FP images. For example, in Fig. E.1(i), a large part of the image was detected as a drone, similar to the Drone-Net model. However, most of the FP detections occurred for no-drone test images. Figure E.1(j), (k), (l), and (m) shows that drone-like shapes could be detected as a drone.

In addition, model 2 detected two FP images, one of them from the no-drone test image set. A noisy image from the no-drone image set was tested, and model 2 detected smoke as a drone. This FP image seen in Fig. E.1(o) can be caused by the shape of the smoke, which appears like a drone in the image.

As observed above for the other models, model 3 also detected objects which are not drones. A drone is detected in a few of the test images from the no-drone test set seen in Fig. E.1(q), (r), (s), and (t). However, there are no drones in these images.

Some of the common FP detections among the models are shown in Fig. E.1(n), (p), and (u). In these figures, it is seen that a drone is detected in one of the test images from the web. However, another object, a commercial aircraft, is detected as a drone, which is a FP instead of the drone to the bottom left of the aircraft.

7.0 CONCLUSIONS

Drone detection is a critical element of counter-drone systems, which also include other subsystems such as drone type classifiers (malicious or friendly) and neutralisation subsystems, such as jamming, laser guns or shotguns. We believe that AI will protect the skies from incoming malicious drone threats. This paper shows that drones can be detected with high accuracy by using powerful available real-time object detection algorithms.

The proposed models are trained by using different kinds of images of drones and compared with three state-of-the-art CNN models for object/drone detection that are available in the public domain and also used as baseline models for transfer learning to the new models. The best models are found to be model 2 and model 3. The results show that drones in AirSim images can be detected with high precision by using pre-trained convolutional layers and training with AirSim and Drone-Net images. Additionally, it is observed that model 1 produces promising results compared with the state-of-the-art Drone-Net drone detection model.

Both state-of-the-art object detection algorithms, i.e., Darknet-53 (model 2) and EfficientNet-B0 (model 3), showed similar results. However, in real-world applications such as counter-drone systems, the object detection method must be operated with limited resources. EfficientNet-B0 provides state-of-the-art accuracy with a neural network that is nine times smaller and consumes significantly less computation power compared with other state-of-the-art object detectors. In future work, the EfficientNet-B0 drone detection model tested here could be integrated as part of our counter-drone system in which, using DRL methods, a guardian drone could detect and counter malicious drones while respecting and

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

avoiding obstacles and legal drones. The accuracy of object detection and the fast response time are very important challenges to track and catch drones. Finally, in the future, the inaccurate bounding boxes of the auto-labelling process must be investigated to obtain more robust drone detection models. One possible solution is to update the simulator when possible, since the updated simulator API might fix the issue relating to the processing time between the image capture and the receipt of the drone flight data from the simulator. A second solution would be to optimise the Yolo object detection function to make the bounding boxes centre correctly.

ACKNOWLEDGEMENTS

This work was funded by the Ministry of Science, Innovation and Universities of Spain under grant no. TRA2019 and the SESAR Joint Undertaking (JU) project CORUS-XUAM, under grant agreement No 101017682. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the SESAR JU members other than the Union.

REFERENCES

1. KRATKY, M. and FARLIK, J. Countering UAVs–the mover of research in military technology, *Defence Sci. J.*, 2018, **68**, (5), pp 460–466.
2. CHOI, B., OH, D., KIM, S., CHONG, J.-W. and LI, Y.-C. Long-range drone detection of 24 g FMCW radar with e-plane sectoral horn array, *Sensors*, 2018, **18**, (12), p 4171.
3. BERNARDINI, A., MANGIATORDI, F., PALLOTTI, E. and CAPODIFERRO, L. Drone detection by acoustic signature identification, *Electron. Imaging*, 2017, **10**, pp 60–64.
4. DE HAAG, M.U., BARTONE, C.G. and BRAASCH, M.S. Flight-test evaluation of small form-factor LiDAR and radar sensors for sUAS detect-and-avoid applications, 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), IEEE, 2016, pp 1–11.
5. SAMARAS, S., DIAMANTIDOU, E., ATALOGLOU, D., SAKELLARIOU, N., VAFEIADIS, A., MAGOULIANITIS, V., LALAS, A., DIMOU, A., ZARPALAS, D., VOTIS, K., DARAS, P. and TZOVARAS, D. Deep learning on multi sensor data for counter UAV applications–a systematic review, *Sensors*, 2019, **19**, (22), p. 4837.
6. Çetin, E., BARRADO, C. and PASTOR, E. Counter a drone in a complex neighborhood area by deep reinforcement learning, *Sensors*, 2020, **20**, (8), p 2320.
7. GIRSHICK, R. Fast R-CNN, Proceedings of the IEEE International Conference on Computer vVision, 2015, pp 1440–1448.
8. REN, S., HE, K., GIRSHICK, R. and SUN, J. Faster R-CNN: Towards real-time object detection with region proposal networks, in *Adv. Neural Inform. Process. Syst.*, 2015, pp 91–99.
9. CHOLLET, F. Xception: Deep learning with depthwise separable convolutions, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp 1251–1258.
10. REDMON, J., DIVVALA, S., GIRSHICK, R. and FARHADI, A. You only look once: Unified, real-time object detection, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp 779–788.
11. TAN, M. and LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks, International Conference on Machine Learning, PMLR, 2019, pp 6105–6114.
12. TAN, M., PANG, R. and LE, Q.V. Efficientdet: Scalable and efficient object detection, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp 10781–10790.
13. SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D. Erhan, D., Vanhoucke, V. and Rabinovich, A. Going deeper with convolutions, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp 1–9.
14. REDMON, J. and FARHADI, A. Yolo9000: Better, faster, stronger, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp 7263–7271.
15. BHATTARAI, N., NAKAMURA, T. and MOZUMDER, C. Real time human detection and localization using consumer grade camera and commercial UAV, 2018.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

16. RUDOL, P. and DOHERTY, P. Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery, 2008 IEEE Aerospace Conference, IEEE, 2008, pp 1–8.
17. YANG, D., YURTSEVER, E., RENGANATHAN, V., REDMILL, K.A. and Özgüner, Ü A vision-based social distance and critical density detection system for covid-19, arXiv preprint arXiv:2007.03578, 2020.
18. Coronavirus disease (covid-19) pandemic. https://www.who.int/emergencies/diseases/novel-coronavirus-2019, last Accessed: 2021-01-10.
19. CRAYE, C. and ARDJOUNE, S. Spatio-temporal semantic segmentation for drone detection, 2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, 2019, pp 1–5.
20. Drone-net, https://github.com/chuanenlin/drone-net note = Last Accessed: 2020-06-02, 2020.
21. XIAOPING, L., SONGZE, L., BOXING, Z., YANHONG, W. and FENG, X. Fast aerial uav detection using improved inter-frame difference and svm, *J. Phys. Conf. Ser.*, **1187**, (3). IOP Publishing, 2019, p 032082.
22. HU, Q., DUAN, Q., MAO, Y., ZHOU, X. and ZHOU, G. Diagonalnet: Confidence diagonal lines for the UAV detection, *IEEJ Trans. Electr. Electron. Eng.*, 2019, **14**, (9), pp 1364–1371.
23. JIN, R., JIANG, J., QI, Y., LIN, D. and SONG, T. Drone detection and pose estimation using relational graph networks, *Sensors*, 2019, **19**, (6), p 1479.
24. CARRIO, A., TORDESILLAS, J., VEMPRALA, S., SARIPALLI, S., CAMPOY, P. and HOW, J.P. "Onboard detection and localization of drones using depth maps," *IEEE Access*, 2020, **8**, pp 30480–30490.
25. WYDER, P.M., CHEN, Y.-S., LASRADO, A.J., PELLES, R.J., KWIATKOWSKI, R., COMAS, E.O., KENNEDY, R., MANGLA, A., HUANG, Z., HU, X., XIONG, Z., AHARONI, T., CHUANG, T.-C. and LIPSON, H. Autonomous drone hunter operating by deep learning and all-onboard computations in gps-denied environments, *PloS One*, 2019, **14**, (11).
26. Çetin, E., Counter a drone, https://github.com/EnderUPC/CounterDrone.git note = Last Accessed: 2021-03-23, 2021.
27. SHAH, S., DEY, D., LOVETT, C. and KAPOOR, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065
28. Unreal engine 4, https://www.unrealengine.com/en-US/what-is-unreal-engine-4, last Accessed: 2019-01-29.
29. REDMON, J. Darknet: Open source neural networks in c. http, pjreddie. com/darknet, vol. 2016, 2013.
30. Yolo-v4 and yolo-v3/v2 for windows and linux. https://github.com/AlexeyAB/darknet note = Last Accessed: 2020-05-27, 2020.
31. REDMON, J. and FARHADI, A. Yolov3: An incremental improvement, arXiv, 2018.
32. BISONG, E. Google colaboratory, in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Springer, 2019, pp 59–64.
33. STEVENS, B.L., LEWIS, F.L. and JOHNSON, E.N. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, John Wiley & Sons, 2015.
34. TAYLOR, M.E. and STONE, P. Transfer learning for reinforcement learning domains: A survey, *J. Mach. Learn. Res.*, 2009, **10**, pp 1633–1685.
35. LIN, T.-Y., MAIRE, M., BELONGIE, S., HAYS, J., PERONA, P., RAMANAN, D., Dollár, P., and ZITNICK, C.L. Microsoft coco: Common objects in context, European Conference on Computer Vision, Springer, 2014, pp 740–755.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

APPENDIX

A.0 Darknet-53

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure A.1. Darknet-53[31].

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

B.0 Drone-Net & Model 1 NN summary

```
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     16       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  16 0.150 BF
   1 max                2x 2/ 2    416 x 416 x  16 ->  208 x 208 x  16 0.003 BF
   2 conv     32       3 x 3/ 1    208 x 208 x  16 ->  208 x 208 x  32 0.399 BF
   3 max                2x 2/ 2    208 x 208 x  32 ->  104 x 104 x  32 0.001 BF
   4 conv     64       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  64 0.399 BF
   5 max                2x 2/ 2    104 x 104 x  64 ->   52 x  52 x  64 0.001 BF
   6 conv    128       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x 128 0.399 BF
   7 max                2x 2/ 2     52 x  52 x 128 ->   26 x  26 x 128 0.000 BF
   8 conv    256       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 256 0.399 BF
   9 max                2x 2/ 2     26 x  26 x 256 ->   13 x  13 x 256 0.000 BF
  10 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  11 max                2x 2/ 1     13 x  13 x 512 ->   13 x  13 x 512 0.000 BF
  12 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  13 conv    256       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 256 0.089 BF
  14 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  15 conv     18       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x  18 0.003 BF
  16 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  17 route  13                                     ->   13 x  13 x 256
  18 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128 0.011 BF
  19 upsample                 2x    13 x  13 x 128 ->   26 x  26 x 128
  20 route  19 8                                   ->   26 x  26 x 384
  21 conv    256       3 x 3/ 1     26 x  26 x 384 ->   26 x  26 x 256 1.196 BF
  22 conv     18       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x  18 0.006 BF
  23 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure B.1. Drone-net and model 1 neural network model summary.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

C.0 Model 2 NN Summary

```
mini_batch = 1, batch = 16, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  19 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  20 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  22 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  23 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  24 Shortcut Layer: 21,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  25 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  26 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  27 Shortcut Layer: 24,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  28 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  29 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  30 Shortcut Layer: 27,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  31 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  32 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  33 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  34 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  35 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  36 Shortcut Layer: 33,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  37 conv    512       3 x 3/ 2     52 x  52 x 256 ->   26 x  26 x 512 1.595 BF
  38 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  39 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  40 Shortcut Layer: 37,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  41 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  42 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  43 Shortcut Layer: 40,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  44 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  45 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  46 Shortcut Layer: 43,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  47 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  48 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  49 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  50 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  51 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  52 Shortcut Layer: 49,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  53 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  54 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  55 Shortcut Layer: 52,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  56 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  57 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  58 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  59 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  60 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  61 Shortcut Layer: 58,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  62 conv   1024       3 x 3/ 2     26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
  63 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  64 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  65 Shortcut Layer: 62,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  66 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  67 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  68 Shortcut Layer: 65,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  69 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  70 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  81 conv     18       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  18 0.006 BF
  82 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  83 route  79                                   ->   13 x  13 x 512
  84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  85 upsample                 2x    13 x  13 x 256 ->   26 x  26 x 256
  86 route  85 61                                ->   26 x  26 x 768
  87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
  88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  93 conv     18       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  18 0.012 BF
  94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  95 route  91                                   ->   26 x  26 x 256
  96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
  97 upsample                 2x    26 x  26 x 128 ->   52 x  52 x 128
  98 route  97 36                                ->   52 x  52 x 384
  99 conv    128       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     18       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  18 0.025 BF
 106 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure C.1. Model 2 neural network summary.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

D.0 Model 3 NN summary

```
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 2    416 x 416 x   3 ->  208 x 208 x  32 0.075 BF
   1 conv     32       1 x 1/ 1    208 x 208 x  32 ->  208 x 208 x  32 0.089 BF
   2 conv     32/  32  3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  32 0.025 BF
   3 avg                           208 x 208 x  32 ->     32
   4 conv      8       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x   8 0.000 BF
   5 conv     32       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x  32 0.000 BF
   6 scale Layer: 2
   7 conv     16       1 x 1/ 1    208 x 208 x  32 ->  208 x 208 x  16 0.044 BF
   8 conv     96       1 x 1/ 1    208 x 208 x  16 ->  208 x 208 x  96 0.133 BF
   9 conv     96/  96  3 x 3/ 2    208 x 208 x  96 ->  104 x 104 x  96 0.019 BF
  10 avg                           104 x 104 x  96 ->     96
  11 conv     16       1 x 1/ 1      1 x   1 x  96 ->    1 x   1 x  16 0.000 BF
  12 conv     96       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x  96 0.000 BF
  13 scale Layer: 9
  14 conv     24       1 x 1/ 1    104 x 104 x  96 ->  104 x 104 x  24 0.050 BF
  15 conv    144       1 x 1/ 1    104 x 104 x  24 ->  104 x 104 x 144 0.075 BF
  16 conv    144/ 144  3 x 3/ 1    104 x 104 x 144 ->  104 x 104 x 144 0.028 BF
  17 avg                           104 x 104 x 144 ->    144
  18 conv      8       1 x 1/ 1      1 x   1 x 144 ->    1 x   1 x   8 0.000 BF
  19 conv    144       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x 144 0.000 BF
  20 scale Layer: 16
  21 conv     24       1 x 1/ 1    104 x 104 x 144 ->  104 x 104 x  24 0.075 BF
  22 dropout    p = 0.000        259584  ->   259584
  23 Shortcut Layer: 14,  wt = 0, wn = 0, outputs: 104 x 104 x  24 0.000 BF
  24 conv    144       1 x 1/ 1    104 x 104 x  24 ->  104 x 104 x 144 0.075 BF
  25 conv    144/ 144  5 x 5/ 2    104 x 104 x 144 ->   52 x  52 x 144 0.019 BF
  26 avg                            52 x  52 x 144 ->    144
  27 conv      8       1 x 1/ 1      1 x   1 x 144 ->    1 x   1 x   8 0.000 BF
  28 conv    144       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x 144 0.000 BF
  29 scale Layer: 25
  30 conv     40       1 x 1/ 1     52 x  52 x 144 ->   52 x  52 x  40 0.031 BF
  31 conv    192       1 x 1/ 1     52 x  52 x  40 ->   52 x  52 x 192 0.042 BF
  32 conv    192/ 192  5 x 5/ 1     52 x  52 x 192 ->   52 x  52 x 192 0.026 BF
  33 avg                           52 x  52 x 192 ->    192
  34 conv     16       1 x 1/ 1      1 x   1 x 192 ->    1 x   1 x  16 0.000 BF
  35 conv    192       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x 192 0.000 BF
  36 scale Layer: 32
  37 conv     40       1 x 1/ 1     52 x  52 x 192 ->   52 x  52 x  40 0.042 BF
  38 dropout    p = 0.000        108160  ->   108160
  39 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x  40 0.000 BF
  40 conv    192       1 x 1/ 1     52 x  52 x  40 ->   52 x  52 x 192 0.042 BF
  41 conv    192/ 192  3 x 3/ 1     52 x  52 x 192 ->   52 x  52 x 192 0.009 BF
  42 avg                           52 x  52 x 192 ->    192
  43 conv     16       1 x 1/ 1      1 x   1 x 192 ->    1 x   1 x  16 0.000 BF
  44 conv    192       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x 192 0.000 BF
  45 scale Layer: 41
  46 conv     80       1 x 1/ 1     52 x  52 x 192 ->   52 x  52 x  80 0.083 BF
  47 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
  48 conv    384/ 384  3 x 3/ 1     52 x  52 x 384 ->   52 x  52 x 384 0.019 BF
  49 avg                           52 x  52 x 384 ->    384
  50 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
  51 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
  52 scale Layer: 48
  53 conv     80       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x  80 0.166 BF
  54 dropout    p = 0.000        216320  ->   216320
  55 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  52 x  52 x  80 0.000 BF
  56 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
  57 conv    384/ 384  3 x 3/ 1     52 x  52 x 384 ->   52 x  52 x 384 0.019 BF
  58 avg                           52 x  52 x 384 ->    384
  59 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
  60 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
  61 scale Layer: 57
  62 conv     80       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x  80 0.166 BF
  63 dropout    p = 0.000        216320  ->   216320
  64 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  52 x  52 x  80 0.000 BF
  65 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
  66 conv    384/ 384  5 x 5/ 2     52 x  52 x 384 ->   26 x  26 x 384 0.013 BF
  67 avg                           26 x  26 x 384 ->    384
  68 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
  69 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
  70 scale Layer: 66
  71 conv    112       1 x 1/ 1     26 x  26 x 384 ->   26 x  26 x 112 0.058 BF
  72 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
  73 conv    576/ 576  5 x 5/ 1     26 x  26 x 576 ->   26 x  26 x 576 0.019 BF
  74 avg                           26 x  26 x 576 ->    576
  75 conv     32       1 x 1/ 1      1 x   1 x 576 ->    1 x   1 x  32 0.000 BF
  76 conv    576       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x 576 0.000 BF
  77 scale Layer: 73
  78 conv    112       1 x 1/ 1     26 x  26 x 576 ->   26 x  26 x 112 0.087 BF
  79 dropout    p = 0.000        75712  ->   75712
  80 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  26 x  26 x 112 0.000 BF
  81 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
  82 conv    576/ 576  5 x 5/ 1     26 x  26 x 576 ->   26 x  26 x 576 0.019 BF
  83 avg                           26 x  26 x 576 ->    576
  84 conv     32       1 x 1/ 1      1 x   1 x 576 ->    1 x   1 x  32 0.000 BF
  85 conv    576       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x 576 0.000 BF
  86 scale Layer: 82
  87 conv    112       1 x 1/ 1     26 x  26 x 576 ->   26 x  26 x 112 0.087 BF
```

Figure D.1. Model 3 neural network summary.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

```
 88 dropout    p = 0.000        75712  ->   75712
 89 Shortcut Layer: 80,  wt = 0, wn = 0, outputs:  26 x  26 x 112 0.000 BF
 90 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
 91 conv    576/ 576  5 x 5/ 2     26 x  26 x 576 ->   13 x  13 x 576 0.005 BF
 92 avg                           13 x  13 x 576 ->    576
 93 conv     32       1 x 1/ 1      1 x   1 x 576 ->     1 x   1 x  32 0.000 BF
 94 conv    576       1 x 1/ 1      1 x   1 x  32 ->     1 x   1 x 576 0.000 BF
 95 scale Layer: 91
 96 conv    192       1 x 1/ 1     13 x  13 x 576 ->    13 x  13 x 192 0.037 BF
 97 conv    960       1 x 1/ 1     13 x  13 x 192 ->    13 x  13 x 960 0.062 BF
 98 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->    13 x  13 x 960 0.008 BF
 99 avg                           13 x  13 x 960 ->     960
100 conv     64       1 x 1/ 1      1 x   1 x 960 ->     1 x   1 x  64 0.000 BF
101 conv    960       1 x 1/ 1      1 x   1 x  64 ->     1 x   1 x 960 0.000 BF
102 scale Layer: 98
103 conv    192       1 x 1/ 1     13 x  13 x 960 ->    13 x  13 x 192 0.062 BF
104 dropout    p = 0.000        32448  ->   32448
105 Shortcut Layer: 96,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
106 conv    960       1 x 1/ 1     13 x  13 x 192 ->    13 x  13 x 960 0.062 BF
107 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->    13 x  13 x 960 0.008 BF
108 avg                          13 x  13 x 960 ->     960
109 conv     64       1 x 1/ 1      1 x   1 x 960 ->     1 x   1 x  64 0.000 BF
110 conv    960       1 x 1/ 1      1 x   1 x  64 ->     1 x   1 x 960 0.000 BF
111 scale Layer: 107
112 conv    192       1 x 1/ 1     13 x  13 x 960 ->    13 x  13 x 192 0.062 BF
113 dropout    p = 0.000        32448  ->   32448
114 Shortcut Layer: 105,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
115 conv    960       1 x 1/ 1     13 x  13 x 192 ->    13 x  13 x 960 0.062 BF
116 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->    13 x  13 x 960 0.008 BF
117 avg                          13 x  13 x 960 ->     960
118 conv     64       1 x 1/ 1      1 x   1 x 960 ->     1 x   1 x  64 0.000 BF
119 conv    960       1 x 1/ 1      1 x   1 x  64 ->     1 x   1 x 960 0.000 BF
120 scale Layer: 116
121 conv    192       1 x 1/ 1     13 x  13 x 960 ->    13 x  13 x 192 0.062 BF
122 dropout    p = 0.000        32448  ->   32448
123 Shortcut Layer: 114,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
124 conv    960       1 x 1/ 1     13 x  13 x 192 ->    13 x  13 x 960 0.062 BF
125 conv    960/ 960  3 x 3/ 1     13 x  13 x 960 ->    13 x  13 x 960 0.003 BF
126 avg                          13 x  13 x 960 ->     960
127 conv     64       1 x 1/ 1      1 x   1 x 960 ->     1 x   1 x  64 0.000 BF
128 conv    960       1 x 1/ 1      1 x   1 x  64 ->     1 x   1 x 960 0.000 BF
129 scale Layer: 125
130 conv    320       1 x 1/ 1     13 x  13 x 960 ->    13 x  13 x 320 0.104 BF
131 conv   1280       1 x 1/ 1     13 x  13 x 320 ->    13 x  13 x1280 0.138 BF
132 conv    256       1 x 1/ 1     13 x  13 x1280 ->    13 x  13 x 256 0.111 BF
133 conv    256       3 x 3/ 1     13 x  13 x 256 ->    13 x  13 x 256 0.199 BF
134 Shortcut Layer: 132,  wt = 0, wn = 0, outputs:  13 x  13 x 256 0.000 BF
135 conv     18       1 x 1/ 1     13 x  13 x 256 ->    13 x  13 x  18 0.002 BF
136 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
137 route  133                                   ->    13 x  13 x 256
138 conv    128       1 x 1/ 1     13 x  13 x 256 ->    13 x  13 x 128 0.011 BF
139 upsample            2x    13 x  13 x 128 ->    26 x  26 x 128
140 Shortcut Layer: 90,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
(  26 x  26 x 128) + (  26 x  26 x 576)
141 conv    128       3 x 3/ 1     26 x  26 x 128 ->    26 x  26 x 128 0.199 BF
142 Shortcut Layer: 139,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
143 Shortcut Layer: 90,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
(  26 x  26 x 128) + (  26 x  26 x 576)
144 conv     18       1 x 1/ 1     26 x  26 x 128 ->    26 x  26 x  18 0.003 BF
145 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure D.1. Continued.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press

E.0 FP detections

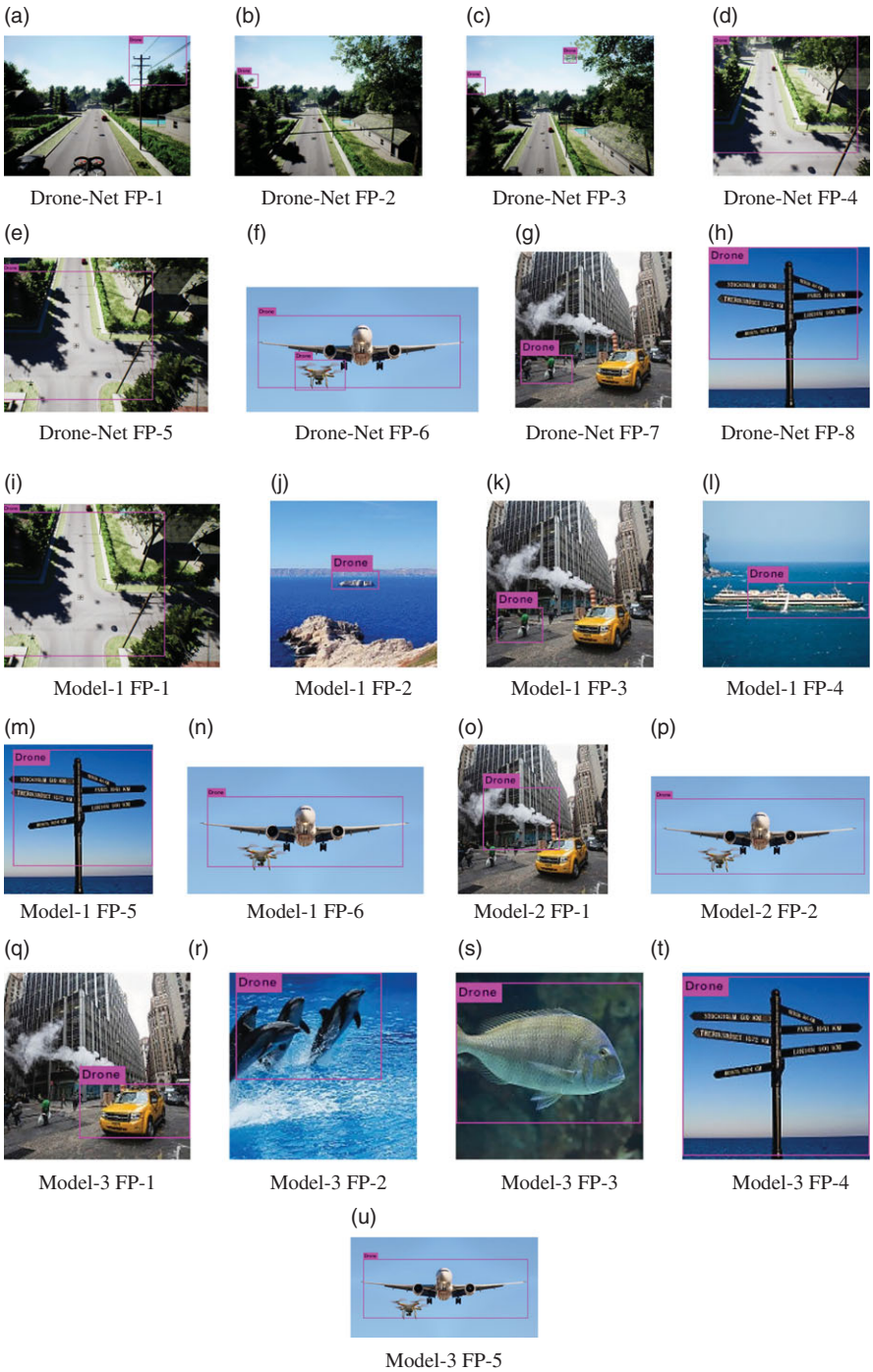

Figure E.1. FP images.

https://doi.org/10.1017/aer.2021.43 Published online by Cambridge University Press