Department of Computer Science

Submitted in part fulfilment for the degree of BSc

Tracking and recognition from videos taken by drones

Ethan Plumbly

15 June 2021

Project Supervisor: Adrian Bors

Supervisor: Lilian Blot

ACKNOWLEDGEMENTS

I would like to acknowledge my brother, for aiding me with all the necessary library installations and set up. I would also like to thank my friends and family for keeping me focused as my final year came to a close.

STATEMENT OF ETHICS

Under UK law, regulations for camera equipped drones state that drones must not be flown: over or within 150 metres of any congested area; over or within 150 metres of an organised open-air assembly of more than 1000 people; or within 50 metres of people and/or property that are not under the direct control of the user. Any pictures taken with the drone must also not break any associated privacy laws.

Since the drone will be used solely indoors to take pictures only of myself, none of these regulations apply and the project is ethically sound.

TABLE OF CONTENTS

Executive summary **8**
1 Introduction **1**
2 Literature Review & Background **2**
2.1 Drone Research **2**
2.2 Previous Works **3**
2.3 Fundamentals of Object Detection **4**
2.4 TensorFlow **5**
3 Design and Implementation **6**
3.1 Drone & Video Processing **6**
3.2 Machine Learning & TensorFlow **7**
3.2.1 Data Collection 7
3.2.2 Data Labelling 8
3.2.3 Model Selection 9
3.2.4 Model Configuration 11
3.2.5 Training 11
3.2.6 TensorBoard 11
3.2.7 Export & Inference 12
3.3 Drone Movement & Autonomous Control **13**
3.3.1 Relative Direction 13
3.3.2 Relative Rotation 15
4 Results & Evaluation **16**
4.1 TensorBoard Graphs **16**
4.1.1 Localisation Loss 16
4.1.2 Classification Loss 18
4.1.3 Regularisation Loss 19
4.1.4 Total Loss 20
4.1.5 Learning Rate 21
4.2 Object Detection Testing **22**
4.2.1 Accuracy 22
4.2.2 Speed 25

4.2.3 Adaptability 26
4.3 Autonomous Control Testing 27
4.3.1 Baseline 27
4.3.2 Close Up 28
4.3.3 Far 29
5 Conclusion 30
Appendix A 31
Bibliography 34

TABLE OF FIGURES

Figure 1.1: System Overview 1
Figure 2.1: Typical Neural Network [40] 4
Figure 2.2: Convolution [40] 5
Figure 2.3: Convolutional Neural Network [40] 5
Figure 3.1: Data Labelling 8
Figure 3.2: MobileNet Architecture [50] 10
Figure 3.3: Faster R-CNN Architecture [51] 10
Figure 3.4: RetinaNet Architecture [52] 11
Figure 3.5: Drone Vision & Calculation Diagram 14
Figure 4.1: MobileNet Localisation Loss 17
Figure 4.2: R-CNN Localisation Loss 17
Figure 4.3: RetinaNet Localisation Loss 17
Figure 4.4: MobileNet Classification Loss 18
Figure 4.5: R-CNN Classification Loss 18
Figure 4.6: RetinaNet Classification Loss 18
Figure 4.7: MobileNet Regularisation Loss 19
Figure 4.8: R-CNN Regularisation graph 19
Figure 4.9: RetinaNet Regularisation Loss 20
Figure 4.10: MobileNet Total Loss 20
Figure 4.11: R-CNN Total Loss 20
Figure 4.12: RetinaNet Total Loss 21
Figure 4.13: MobileNet Accuracy Test (Standard) 22
Figure 4.14: R-CNN Accuracy Test (Standard) 22
Figure 4.15: RetinaNet Accuracy Test (Standard) 23
Figure 4.16: MobileNet Accuracy Test (Motion Blur) 23
Figure 4.17: R-CNN Accuracy Test (Motion Blur) 23
Figure 4.18: RetinaNet Accuracy Test (Motion Blur) 24
Figure 4.19: MobileNet Accuracy Test (Glitch) 24
Figure 4.20: R-CNN Accuracy Test (Glitch) 24
Figure 4.21: RetinaNet Accuracy Test (Glitch) 25
Figure 4.22: Adaptability Test Image 27
Figure 4.23: Autonomous Control Test Image (Baseline) 27
Figure 4.24: Autonomous Control Test Image (Close Up) 28
Figure 4.25: Autonomous Control Test Image (Far) 29

TABLE OF TABLES

Table 1: Raw Training Data 8
Table 2: TensorFlow Model Zoo 31

Executive summary

Autonomous drone research has been a key area of innovation in computer vision in recent years. The following report details the research and implementation of an autonomous Tello drone system that extracts and displays a live camera feed that is then processed using machine learning techniques through TensorFlow. Open palm gestures are localised and classified within each frame and displayed using bounding box overlays. The coordinates for these boxes are then used to determine the boxes relative position within the image such that orthogonal velocities can be calculated that will centre the gesture on the screen. Once these values are calculated, the velocities are transmitted back to the done using an interface library resulting in corresponding real-world movement. This results in an autonomous system that allows the user to move the drone without directly controlling it.

In total, three TensorFlow network architectures are trained using a custom data set that I collected and formatted myself. This dataset was collected within a single room under varying lighting conditions to diversify the data. The drone was suspended from varying positions and several angles of the open palm gesture were recorded to ensure as many scenarios are represented as possible. The data is then labelled using LabelImg software and converted to TFRecords to be trained by the aforementioned TensorFlow models. Models are specifically chosen to cover a diverse range of accuracies and speed attributes. The lightweight MobileNet priorities speed over accuracy, the middleweight R-CNN balances accuracy and speed equally, and the heavyweight RetinaNet prioritises accuracy over speed.

Once training is complete for each network (or ended early for the RetinaNet due to taking too long), a vigorous evaluation process is undertaken that analyses each network's training through TensorBoard, TensorFlow's visualisation toolkit. This allowed several properties of the training process to be studied such as localisation loss, classification loss and learning rate. Practical tests are also performed that showcase various scenarios that the drone might encounter as well as some possible visual glitches that occur due to misconfiguration. Surprisingly, the RetinaNet ended up performing the worst while the middleweight Faster R-CNN achieved the highest accuracy scores. Unfortunately, due to the longer processing speed, the R-CNN is deemed unsuitable for processing live video, so the still relatively efficient MobileNet is left as the only appropriate architecture.

1 Introduction

Computer Vision has become very prevalent in recent times with the development of machine learning and artificial intelligence technologies used in self-driving cars and general footage analysis. Drone research has many industrial applications such as package delivery, landscape detailing and general reconnaissance. Automated drones, that could react to stimuli captured from an on-board camera independently, are especially beneficial as it reduces the need for human operators. The following project report will detail my research and findings around drone technology and machine learning techniques. My goal is to train an object detection model capable of detecting an open palm hand gesture in live footage that can be tracked physically via an automated drone control system. As such there are three main stages to this project:

1. Extracting live video footage from a small indoor drone that can be interacted with through a python library.
2. Training a model to detect the open palm gesture which includes deciding on and testing various models and collecting training data.
3. Writing an automated algorithm that extracts the features of the detected gesture (distance, width, height etc.) and controls the drone to automatically follow the hand gesture.

Once each stage is completed, I will hopefully have an automated system that will process and follow an open palm gesture localised within live drone footage. In this sense, the system allows the user to move the drone to any desired position to capture a scene without directly controlling the drone. **Figure 1.1** diagrams the system overview.

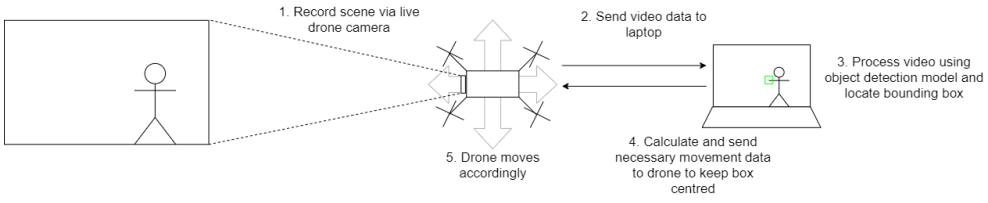

***Figure 1.1**: System Overview*

2 Literature Review & Background

2.1 Drone Research

Object recognising drone research has many real-world applications [1]. For example, the area of agriculture requires the monitoring and travel of large acres of land that would usually take a human agent long periods of time, but a drone would be able to cover the same distance quickly and with ease. This allows farmers to monitor crop diseases and yield prediction [2] [3]. Asset inspecting drones can search for module faults in solar panel farms [4] and turbine farms [5] [6] [7] as well as inspecting bridges [8] [9] [10], oil refineries [11] [12], and railways [13] [14] [15]. Drones and computer vision can also be used in land surveying [16] [17] and site management in the field of construction [18] as well as claim validation in insurance [1].

The following report focuses specifically on live subject tracking that involves the detection and autonomous following of a specified object or subject. Good examples of real-world applications of this are in rescue and security. With rescue, drones can be deployed to search large areas for people of interest and track them once located making it easier for rescuers to reach their location [19] [20]. In security, drones can be used to survey crime scenes [21] and track down criminals to be apprehended [22]. Drones have a key advantage over humans since they are lighter, can move through areas easier, and produce less noise. In a similar situation, drones can be used to track wildlife in large preserves for animal conservation [23] [24] [25], or even sharks [26] and other aquatic life in shallow waters. This can make coastal areas safer as sharks and dangerous animals can be spotted and tracked to be avoided [27]. Drones are especially advantageous in this aspect due to there being less obstacles over water that would pose a danger to the drone's electronics. The affordability and simplicity of drone take-off makes the dispatch of human operators on large boats unnecessary as the latter would be expensive in cost and time. In this sense, the drones also pose less danger to wildlife as a non-intrusive reconnaissance unit that produce little effect on an animal's livelihood, especially with quieter, smaller drones. This also links back to the use of drones in search and rescue of humans in need but in coastal areas, since drones would be able to cover large sections of water efficiently. Minimal processing techniques would also be needed as the detection models would be processing images of water where subjects of interest would be more likely to stick out so only thermal information would be needed [28].

A likely primary application of tracking drones in the real world, is in the film industry as autonomous cameramen [29]. Drones that require no supports can be automated to follow actors while filming with little input from human controllers [30]. This would save on expenses when

it comes to laying camera tracks and having to rely on human reaction time to keep objects and subjects in frame. Mass produced and affordable drones would also make home films easier since no other operator is needed. A single person can use a drone to film themselves. For example, a skater can record their own lines and tricks on their own with no team of camera operators [31].

2.2 Previous Works

In a project similar to the one I have proposed, GitHub user geaxgx1 implemented algorithms for the Tello drone that would act as an autonomous selfie air stick [32]. He did this through a different open-source python repository [33] and a selection of person trackers. In his project, he used a collection of OpenCV's built-in face detectors, but these failed under extreme lighting changes. To solve this, OpenPose [34] was used which detected and tracked the movement of the user's limbs and body language. This worked under harsh lighting and allowed him to always be able to calculate approximately where his head was. With this information, he could estimate the head's position in relation to the centre of the screen which could then be used to control the drone's behaviour using the drone interaction repository. For the actual drone movement, the codebase from another object tracking project [35] is used which was trained to follow a red ball. This project was one of the main inspirations for my project, but I instead decided to train my own machine learning model rather than using a pretrained package.

Another project [36] that I looked at involved ID badges worn by a user that would be detected by a Parrot drone. Each badge contained two components. The first would be used to localise the user within the drones view and the second would be linked to pre-set movement procedures that would control the drone. This project focused less on a machine learning approach, instead choosing to manually detect badges using bottom-up and top-down saliency mechanisms, a novel image similarity measure, and an affine validation procedure. This was another key inspiration for my project, but I instead opted to train a convolutional network to recognise the user and chose to recognise a user's hand instead of a badge. Perhaps a different object such as the badges used in their project would be more distinct in a scene and would make detection simpler, but I wanted a common object that every user would have (a body part such as a hand being an extreme example of this).

Of the current available research in drone-based gesture recognition, two specific examples [37] [38] utilised a LEAP motion controller and its readings as a method of drone control. The dataset consequently consisted of four-dimensional spatiotemporal data that logged the captured motion. One of these examples [38] specifically used a deep

learning approach to determine the corresponding movement sequence and managed to achieve upwards of 90% accuracy on both scaled and unscaled testing datasets. My project aims to instead detect gestures through visual frame data alone and uses a stationary signal approach as opposed to spatiotemporal i.e., non-moving hand signs such as an open palm.

Another paper [39] undertakes a computer vision approach to gesture recognition and describes a three-stage process that undergoes image segregation, object classification, and reactionary movement which is similar to my projected work process. A feature-based classifier determines the presence of five possible gestures that control a set of typical drone actions such as take off and landing. My project intends to differ from this by performing a tracking process that follows the gesture rather than executing prebuilt movement sequences.

2.3 Fundamentals of Object Detection

Objects in video can be detected through many forms, one of which is through machine learning. We have already looked at badge processing methods used in previous projects [36] but the machine learning approach uses a specific artificial intelligence model called a convolutional neural network. A neural network is simply a mathematical model that mimics the structure of a biological sensitivity system. An input sends signals through a network of connected layers to produce an output signal that acts as a reaction to the stimuli.

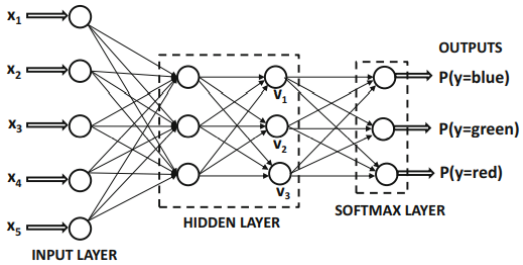

***Figure 2.1**: Typical Neural Network [40]*

A convolutional neural network is a special type of neural net that uses a variation of multi-layered nodes to perform object detection. Its primary operation, convolution, defines a $K = k \times k$ kernel that is used to detect and locate specific regions of interest within an image, in our case, objects and gestures. The filter is applied algorithmically over the pixel data of the source image I to produce the convolved image $I * K$. This convolution filter is the fundamental basis of the convolutional network. The full convolutional network will contain a layer of these convolution filters that are later adjusted in machine

learning to detect specified elements contained within the image. Each layer processes the output of the previous layer.

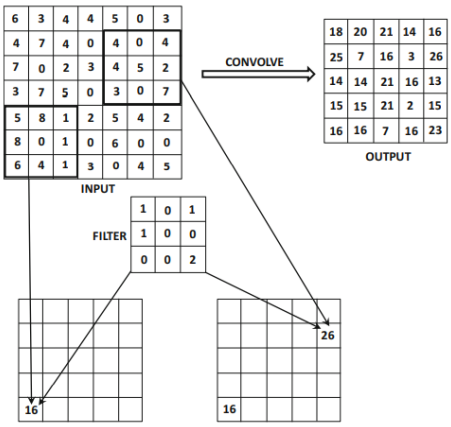

***Figure 2.2**: Convolution [40]*

Once an input has been convoluted, the results are pooled and rectified to be used as further inputs in later network operations. The process of convolution can happen multiple times until a fully connected output is reached. This output layer contains all the possible classifications for the detected object.

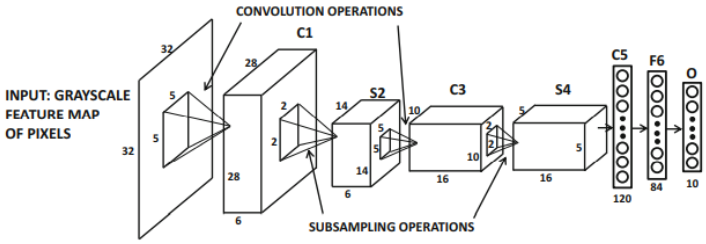

***Figure 2.3**: Convolutional Neural Network [40]*

2.4 TensorFlow

Many deep learning frameworks exist (such as Keras and pyTorch) but for the purposes of this project, the machine learning library used for this project is TensorFlow [41]. The driving philosophy behind TensorFlow is its use of a data structure called a Tensor that is used to represent numerical flow graphs and models along with the appropriate operations. These multi-dimensional arrays make processing images far easier since pixel data would otherwise have to be flattened out manually as an input layer for the neural network. One key feature is the inclusion of TensorBoard, a piece of visualisation software that allows us to evaluate a network's training and performance in real time. This will be paramount in the testing and evaluation section of the report.

3 Design and Implementation

3.1 Drone & Video Processing

A key decision for the project was the choice of drone. Originally, I had hoped to use the robotics research lab at the university but due to COVID restrictions, this would not be possible, so I was left to either collect drone video data online or purchase a drone for myself. For the purposes of this project, I decided the latter would be more suitable to make data collection and real-time testing possible and easier. This meant I would have to purchase a small, affordable, indoor drone with an existing framework for extracting live video at a reasonable resolution. After a lot of deliberation between multiple drones, I decided on the DJI Tello Ryze [42]; a miniature indoor drone capable of 720p transmission in a 100m range.

The drone itself is popular in many institutions due to its affordability and community support. Typically, most drones would be controlled through a physical controller or through a mobile phone app but because I need to be able to process the live footage, a python programmable drone compatible with my laptop and such as the DJI Tello Ryze was necessary. Many popular repositories also exist that provide easy to use modules that interact with the drone in real-time to extract footage and control movement. Both functionalities are paramount for my project since the drone needs to react and follow the detected hand gesture. For the purposes of this project, the DJITelloPy repository [43] is used to communicate with the drone. Many other repositories were researched until this one was decided due to the ease of use and simple controls that would make drone control much simpler.

```
drone = Tello()
drone.connect()
drone.streamoff()
drone.streamon()
capture = drone.get_frame_read()
```

At the start of the program, a Tello object is instantiated that will be used to interact with the physical drone and retrieve data packets. In order for the connection to go through, the processing device (in my case, my laptop) has to connect to a local WiFi network which is transmitted by the physical drone. Once the connection is complete, control functions can be transmitted to and from the drone during runtime which allows us to retrieve visual data and define a capture to be displayed.

To display the footage of the drone so any detected objects can be visualised, OpenCV [44] is used. OpenCV is a library of real-time computer vision functions originally developed by Intel that can be used to display various video inputs for machine learning purposes. This is our primary way of viewing the models live processing and decision making. After any image processing, the finalised image frame can be displayed with the following listing:

```
image_np = capture.frame

cv2.imshow('object_detection', cv2.resize(image_np, (800, 600))
)
if cv2.waitKey(20) & 0xFF == ord('d'):
      cap.release()
      cv2.destroyAllWindows()
      break
```

The purpose of the if statement is for when we want to finish visualising the networks' processing and will break the ever-continuing True loop which retrieves a new frame every iteration.

3.2 Machine Learning & TensorFlow

The main section of the project is the machine learning approach for object detection. This makes heavy use of the TensorFlow library for python which contains some very useful research models that can be cloned from their GitHub [41].

3.2.1 Data Collection

The first step is to collect training data for the models. All video processing and collection will be constrained to depict a single workspace since the requirements of this project warrant no need for a drone that can be used universally. Nevertheless, we want to collect a diverse set of training data that can represent the workspace at several angles and under several lighting conditions. Due to the ongoing pandemic, data collection could only be undertaken by a single person, me, and would require myself to both control the drone and be in the frame simultaneously as the subject. To solve this issue, the drone was suspended at various positions across the room facing myself while I held out my left hand in an open palm gesture. While doing this, I would take pictures through the drone via a mobile device in my left hand. I would then walk around the room with my left hand extended as I performed the open palm gesture to ensure every angle is covered. I would also alter the brightness of the scenes through three separate light sources: sunlight through an open window, a table side lamp, and the overhead ceiling light. The latter two are easy to

adjust as I could simply switch the lights on and off being sure to achieve every combination. To alter the sunlight intake, I would simply repeat the data collection at several times of day. I even used the handheld table-side lamp to brighten the hand gesture at multiple lighting angles which helped diversify the data even more. An example training image can be seen in **Figure 3.1**.

3.2.2 Data Labelling

To label the data, I would then use LabelImg [45], a graphical image annotation tool. I would simply load in the image directory containing my training data and then label the hand gestures with a rectangular tag box. Since I am only hoping to detect a single gesture, I only used a single tag, open_palm. This then made data labelling very simple and streamlined as I could navigate picture to picture for a single tag.

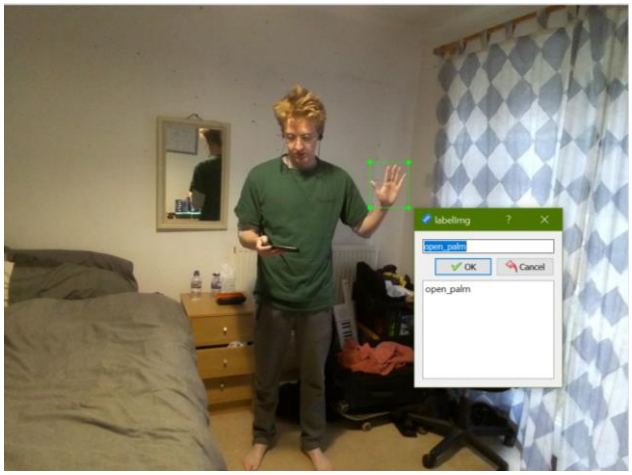

***Figure 3.1**: Data Labelling*

Once every picture has been labelled, the images are separated out into training and testing sets. Using Gilbert's [46] conversion files [47] [48], we can effectively prepare the training and testing files for training as a TFRecord. TFRecords are simple formats containing sequences of binary records that can be used by the machine learning models found in the TensorFlow zoo. A section of the raw information provided to the networks is shown in **Table 1**.

***Table 1**: Raw Training Data*

filename	width	height	class	xmin	ymin	xmax	ymax
1620333340136.png	2592	1936	open_palm	1561	675	1825	914
1620333342564.png	2592	1936	open_palm	1763	867	1952	1020
1620333345152.png	2592	1936	open_palm	1442	546	1812	1003
1620333347223.png	2592	1936	open_palm	491	660	876	1086
1620333349684.png	2592	1936	open_palm	2012	1060	2467	1384

3.2.3 Model Selection

TensorFlow 2 was the version used for training and choosing the machine learning models that I would later evaluate. The TensorFlow team have provided an online zoo containing a number of different model architectures for users to configure for custom training which I have listed in the appendices as **Table 2**. The chosen models have been highlighted.

Each model is classified by two key descriptors, speed and COCO mAP. Speed represents how fast it takes to identify an object in milliseconds within a given frame. COCO mAP (mean average precision) represents how accurate the network is at identifying objects. Networks with a high COCO mAP and a low speed will be able to identify object consistently at the expense of processing speed while a network with high speed and low COCO mAP will have faster processing at the expense of accuracy. A third descriptor is also used to describe the visualisation methods such as keypoints and masks. For the purposes of this project, we will be choosing networks that output bounding boxes for the detected gesture since the box's attributes will be used to determine the motion of the drone in the next section. Three neural network architectures are tested:

1. SSD MobileNet V2 FPNLite 320x320
2. Faster R-CNN ResNet50 V1 640x640
3. SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)

I have purposely chosen to evaluate a diverse range of networks that span different speeds and effectiveness to be sure that an optimum network is chosen. While processing live video, it can be assumed that a faster network is preferable, but I have decided to still train slower networks for the benefit of their accuracy.

The MobileNet network is a Single-Shot multi box Detection network intended to perform object detection in mobile devices and embedded vision applications. It was released in 2017 and uses proven depth-wise separable convolutions to build efficient lightweight neural networks [49]. The 320x320 denotes the size in which images are downscaled to during processing. Smaller image sizes will ensure faster processing at the cost of object resolution and accuracy.

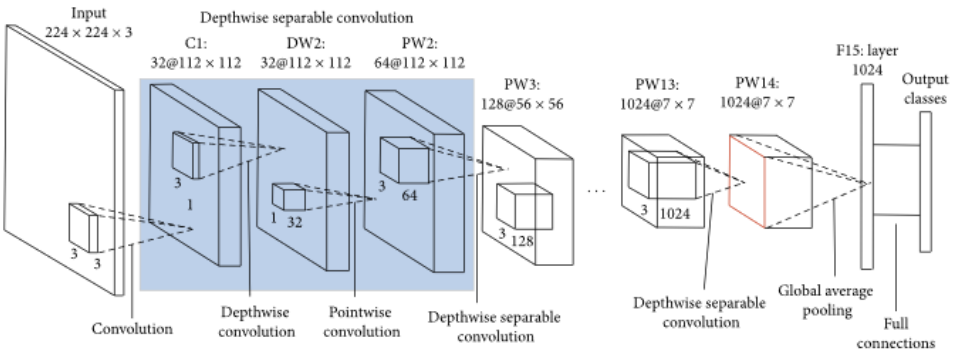

***Figure 3.2**: MobileNet Architecture [50]*

The Faster R-CNN network uses a region based convolutional neural net that proposes regions within the image during the first stage of processing followed by a second stage that identifies the object within the region and produces bounding boxes [49]. In this TensorFlow model however, images are converted to a 640x640 image during processing which would imply greater accuracy during model inference.

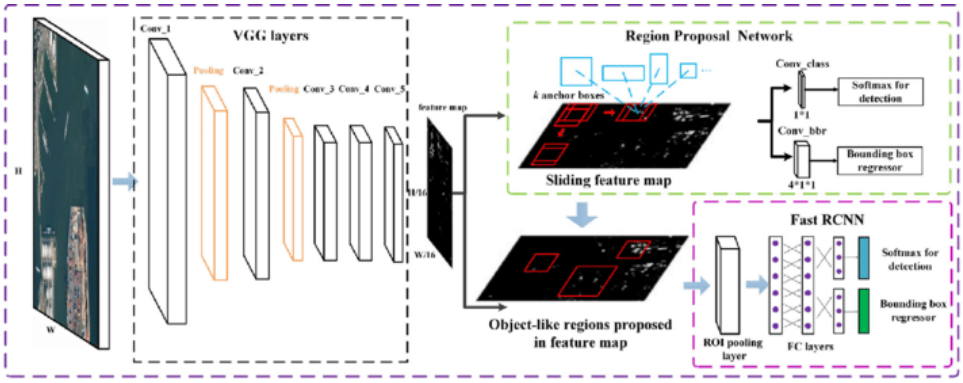

***Figure 3.3**: Faster R-CNN Architecture [51]*

The RetinaNet network was developed by Facebook and uses a Feature Pyramid Network (FPN) backbone on top of a typical feedforward ResNet architecture to generate a rich multiscale convolutional feature pyramid [49]. It differs from the Faster R-CNN by using a single staged network while the latter uses two stages during processing. Its usefulness in live videos is pretty unlikely but I wanted to maintain network diversity and have a network that could confidently detect the gestures in the event the other two could not. With this network, images are downscaled to a hefty 1024x1024 which should provide clear images of the hand gestures to ensure effective training (at the expense of training duration).

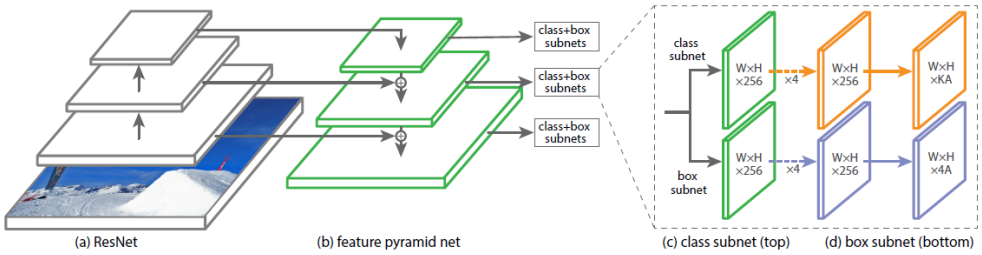

***Figure 3.4**: RetinaNet Architecture [52]*

Other models were selected for use such as an EfficientNet topology that was used in Gilbert's walkthrough [46] but this led to absurdly high loss values and sporadic graph fluctuations that made feasible object detection impossible. This worked out for the best in the end however as it meant I could choose and evaluate a set of network topologies unique from previous projects.

3.2.4 Model Configuration

Before we begin training any model, we need to configure the training files to ensure the training is appropriate. By default, the networks are used for classification, so we change this field to detection since we are only looking to detect a single object class (open_palm). We also update the training and testing data file paths as well as lowering the batch size. Batch size determines the number of samples (images) processed at a time. We lower this because I am undertaking training on a low-performance laptop. We also download the checkpoint files from the model zoo which contain the initial parameters for training. Once we have done all of this, we can start training.

3.2.5 Training

To train our configured model, we then run the main training file (model_main_tf2.py from the cloned TensorFlow directory) while passing in the location of the training data and the modified configuration file.

3.2.6 TensorBoard

During training, we can monitor the network's current performance through TensorBoard, TensorFlow's visualisation toolkit. Here we can evaluate many factors attributed to the network's training such as various loss values (classification, localisation, and regularisation) and general training statistics like learning rates. We can also preview the images being processed to ensure that everything is running smoothly. These will our key source of network evaluation along with experimental test footage.

3.2.7 Export & Inference

Once training is complete, we can then export the finished inference graph that represents the trained network. The code to do this is adapted from Gilbert's inference functions that run a given model on a given image. First, we define the file paths of a specified model and a corresponding label map that lists the class labels to be outputted.

```
model = load_model('…')
category_index = label_map_util.create_category_index_from_labe
lmap(…/label_map.pbtxt', use_display_name=True)
```

Once file locations have been initialised and models are loaded in, we can begin inference of the models by calling an inference function while passing in the model, label map, and capture source (drone camera). During testing, captures could vary from pre-filmed videos or even using a live webcam feed to mimic the drone's point of view without having to connect to the drone itself.

```
run_inference(model, category_index, capture)
```

This run_inference function is responsible for the majority of the code base and contains the core loop that processes live video. Every iteration of the loop, a frame is taken from the drone camera's capture to be processed in another function that runs the actual inference on the single image.

```
ret, image_np = capture.read()
output_dict = run_inference_for_single_image(model, image_np)
```

Within this run_inference_for_single_image() function, images are mathematically transformed using NumPy and TensorFlow so that they are the correctly formatted for use in the network as a tensor.

```
image = np.asarray(image)
input_tensor = tf.convert_to_tensor(image)
input_tensor = input_tensor[tf.newaxis,...]
```

Then this input tensor is submitted to the specified model and outputs are calculated. This output is formatted as a dictionary that can be accessed to infer the location data of any detected objects.

```
    output_dict = model(input_tensor)
    num_detections = int(output_dict.pop('num_detections'))
```

```
    output_dict = {key: value[0, :num_detections].numpy()
                   for key, value in output_dict.items()}
    output_dict['num_detections'] = num_detections
    output_dict['detection_classes'] = output_dict['detection_c
lasses'].astype(np.int64)
    return output_dict
```

With this output, we can then visualise the bounding box components as a OpenCV overlay:

```
vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks_reframed'
, None),
      use_normalized_coordinates=True,
      line_thickness=8)
```

3.3 Drone Movement & Autonomous Control

3.3.1 Relative Direction

When a bounding box has been overlayed over an OpenCV image output, we can retrieve the box parameters using the following listing:

```
boxes = output_dict['detection_boxes']
ymin = boxes[0][0]*height
xmin = boxes[0][1]*width
ymax = boxes[0][2]*height
xmax = boxes[0][3]*width

boxWidth = xmax-xmin
boxHeight = ymax-ymin
boxX = (xmax+xmin)*0.5
boxY = (ymax+ymin)*0.5
boxArea = boxWidth*boxHeight
```

The box parameters are stored in the bounding box dictionary output_dict['detection_boxes'] which we then define as “boxes”. This stores every bounding box that has been located within the frame in order of certainty. Since we have modified the configuration file to only detect, at most, one gesture in a given frame, this would return a list of, at most, one element. This element contains the location data of the box in relation to the size of the image. By multiplying the location data by either width and height, we can infer the pixel coordinates on

which the box exists in frame. These coordinates are defined as ymin, xmin, ymax, xmax.

With those values we can calculate the length of each side and subsequently, the area of the box that contains the gesture. We also calculate the midpoint of the bounding lines in both axes to find the midpoint of the bounding box. Note that in OpenCV, the coordinate system starts in the top left as opposed to an orthodox bottom left system.

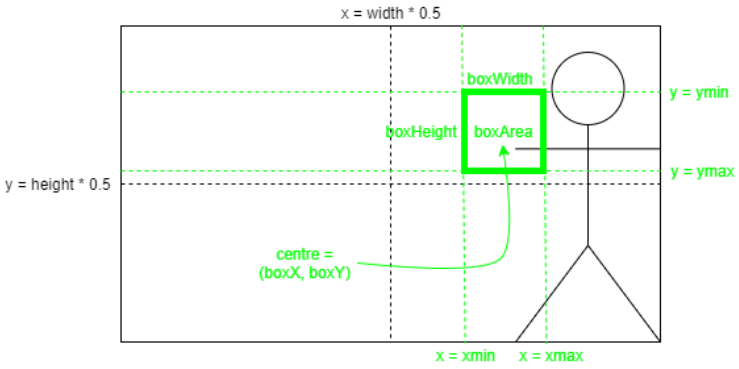

***Figure 3.5**: Drone Vision & Calculation Diagram*

As stated before the DJITelloPy library [43] has been used to interact with the drone and provides the necessary movement commands. These commands allow us to transmit velocity values, with a maximum magnitude of 100, that 'nudge' the drone in the appropriate directions. The idea behind the automated control is to keep the detected gesture in the centre of the capture image so we utilise our calculated position variables to determine the appropriate movements.

```
        xSpeed = (boxX - width / 2) / 20
        ySpeed = (height / 2 - boxY) / 20
        zSpeed = (450 - math.sqrt(boxArea)) / 10
        drone.send_rc_control(xSpeed, zSpeed, ySpeed, 0)
```

Using the calculated centre point coordinates (boxX, boxY), we can determine where the box is on the screen and then manipulate the orthogonal velocities of the drone such that the bounding box is always being translated towards the centre of the drone video. The centre coordinates of the image pane can be found by halving the width and height of the drone footage which can then be compared to the bounding box centre coordinates. The difference is calculated and defines the rightward velocity of the drone. These raw velocities are far too big, sometimes being upwards of 1000, so the results are mediated by dividing by 20.

To manage the depth distance to the gesture, a different method is required since depth does not rely on the gesture box position. Instead, we make use of the area of the box since this is directly proportional to the square of the gesture's distance to the drone camera. After initial testing, a value of 450 was found to be an ideal square rooted area so the difference of the square rooted box area to this, is used to define the forward velocity. This value is then mediated by dividing by 10.

Once all the directional velocities are defined, they are outputted back to the drone as remote-control instructions. This will update the onboard processing of the drone to fly with whatever speed parameters are provided.

3.3.2 Relative Rotation

An effort was also made to implement an algorithm that could calculate the angle of approach of the open palm gesture and adjust the position of the drone appropriately but due to time constraints, this feature was cut from the final product. Regardless however, I will describe the approach that would have been taken. The only information that is provided during model inference, is the bounding box position. At angles of approach that are not direct, the bounding box would appear slimmer and more rectangular. This problem can be modelled as a simple geometric triangle if we wanted to calculate the exact movements necessary but the movements in this project are only performed heuristically so a simplified approach is used instead.

We only need to slightly rotate the drone in the correct direction and the other movement procedures will correct the positioning appropriately as the open palm would now be off centre. An issue, however, is that we cannot infer the correct direction of rotation from a still frame. We will need to take two samples of the bounding box shape at two points in time (likely by rotating a low angle both clockwise and anticlockwise and storing the respective box widths). These values can then be compared to find the best direction to rotate the drone (the direction with the largest box width would lead to a more direct open palm). A greater rotation speed could then be set to orient the drone appropriately. However, the processing steps required to determine the correct direction of rotation would bloat the update function and cause a far slower update cycle which already has to process an object detection model every frame. Because of this setback, the ability to rotate to face the gesture was removed.

4 Results & Evaluation

4.1 TensorBoard Graphs

First, we will evaluate the training of each chosen neural network architecture using TensorBoard, TensorFlow's visualisation toolkit. The primary statistics we will evaluate are localisation loss (ability to find objects locations in the image), classification loss (ability to classify the detected object), regularisation loss (ability to generalise the solutions), and total loss (a combination of all the previously stated loss functions). Each graph in this section plots these statistics against the training steps and is displayed as a time series. We will also assess the learning rate of the training although this will not be as paramount since we will only be using the finished inference graph (although faster training would be preferred).

There are additional statistics to some models, for example, the Faster R-CNN architecture records the objectness loss (ability to detect general object-like subjects), but this statistic is avoided in this report since neither of the other models share it. Due to the sporadic fluctuations that occur epoch to epoch, graph lines have had a smoothing applied for readability's sake. The original lines are depicted at a lower opacity behind the smoothed replacement. Each network's statistics will be examined sequentially and simultaneously to make comparisons easy. Note that the individual model configurations have had a majority of their default settings left unchanged so the number of steps during training will differ. Also note that the RetinaNet's training has been cut short since it was taking upwards of twenty days to complete. This is unsuitable for the timeframes of this project so only 66k of the 100k steps available to the RetinaNet were completed.

4.1.1 Localisation Loss

The localisation loss of an object detection model evaluates its bounding box offset prediction i.e. its ability to locate and pinpoint the position of an object within the image frame.

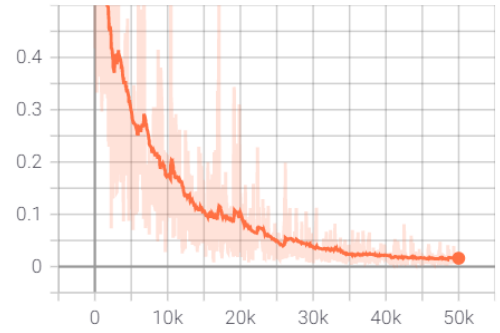

***Figure 4.1**: MobileNet Localisation Loss*

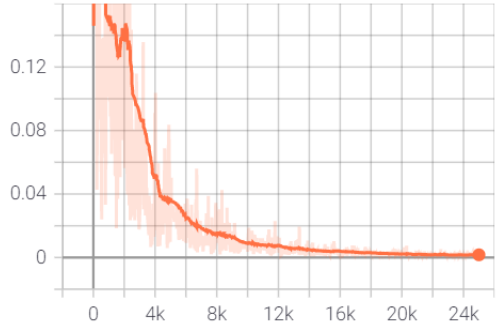

***Figure 4.2**: R-CNN Localisation Loss*

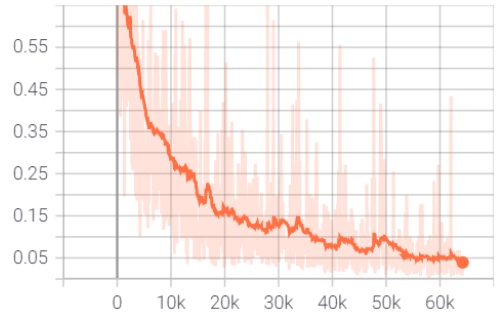

***Figure 4.3**: RetinaNet Localisation Loss*

Initial observations inform us that all graphs undergo a signature logarithmic decline that is common in loss minimisation graphs. The RetinaNet architecture appears to be less consistent than the others even later in training as shown by the largely fluctuating raw data behind the smoothed line in **Figure 4.3**. The other models show fewer outliers and an ongoing 'squeeze' in comparison. An observation could also be made about the R-CNN graph (**Figure 4.2**) with the stagnation during the first 2k steps in training. The other graphs appear to display a more gradual decline although this could possibly be due to a lower number of training steps. Eventually each respective model plateaus and achieves final (smoothed) values of $1.6x10^{-2}$, $1.7x10^{-3}$, and $3.9x10^{-2}$. Surprisingly, the larger and more complex RetinaNet ended

up performing worse than the midground R-CNN. This might be due to the RetinaNet's training being cut short but even with more than half of the training cycle complete, it failed to come close so this is unlikely to be the reason. Perhaps the architecture and techniques of the RetinaNet make it more unsuitable than the others in this specific scenario.

4.1.2 Classification Loss

The classification loss of an object detection model evaluates its ability to classify a detected object correctly once it has been bounded.

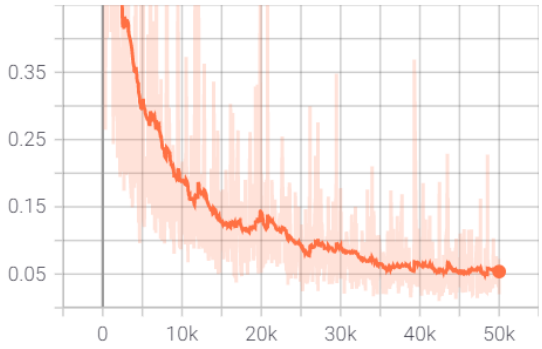

***Figure 4.4**: MobileNet Classification Loss*

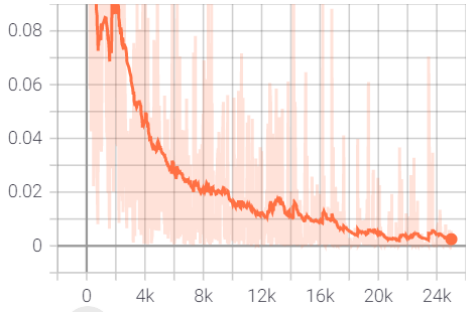

***Figure 4.5**: R-CNN Classification Loss*

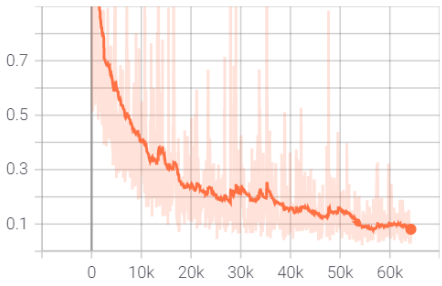

***Figure 4.6**: RetinaNet Classification Loss*

The classification statistics appear far more sporadic in every case when compared to the other loss functions except for the RetinaNet

(**Figure 4.6**). This is clearly shown by the frequent outliers present in each graph when compared to the other statistics. The R-CNN graph (**Figure 4.5**) shows some initial oscillations, and the initial decline shape is delayed for the first 2k steps until the sudden drop forms. The model does appear to achieve very low losses even in the early stages of training as shown by the raw data values on the axes, but these early successes are complemented by equal failures that occur above the smoothened line since otherwise, the final line would be lower. The graph on a whole does also appear more jagged and intermittent throughout training when compared to the others. The final values achieved are $5.4x10^{-2}$, $2.5x10^{-3}$, and $8.1\,x10^{-2}$. Yet again, the RetinaNet shows an unexpected inability to make accurate predictions, this time with classification.

4.1.3 Regularisation Loss

Regularisation is a method not found in every model's training that involves adding information in an attempt to reduce the chance of overfitting.

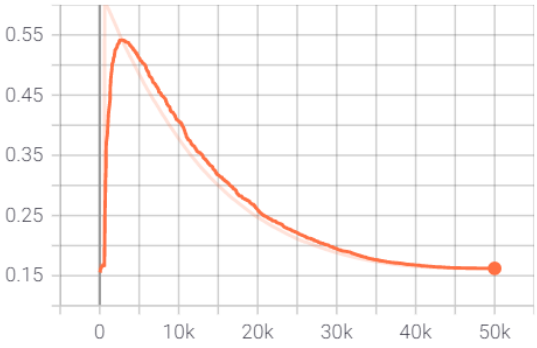

***Figure 4.7**: MobileNet Regularisation Loss*

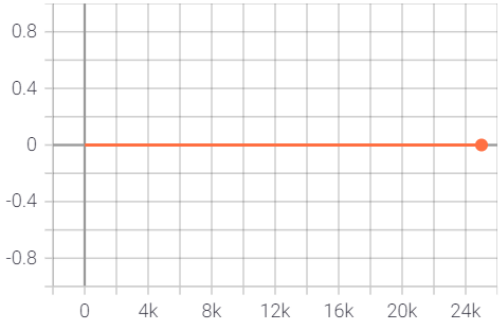

***Figure 4.8**: R-CNN Regularisation graph*

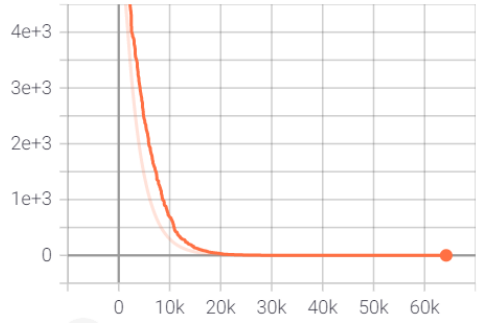

***Figure 4.9**: RetinaNet Regularisation Loss*

As is immediately apparent, only the MobileNet and RetinaNet use regularisation techniques. This is likely due to the greater number of steps, so the danger of overfitting data is more likely. Between the two graphs themselves, the RetinaNet graph (**Figure 4.9**) achieved a plateau far faster than its MobileNet counterpart (**Figure 4.7**) but the raw values themselves go upwards of 10^3 especially during initial states. The final values reached by the MobileNet and RetinaNet are $1.6x10^{-1}$ and $1.4x10^{-1}$ respectively so the RetinaNet did end up achieving a better accuracy score in regularisation.

4.1.4 Total Loss

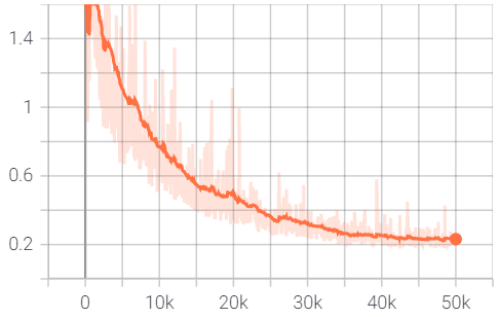

***Figure 4.10**: MobileNet Total Loss*

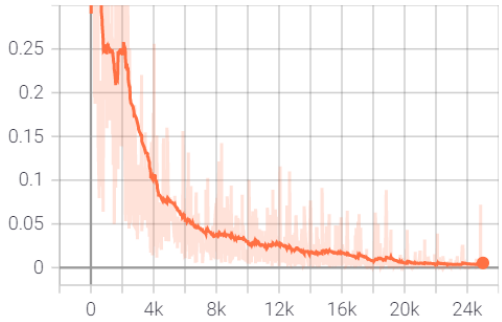

***Figure 4.11**: R-CNN Total Loss*

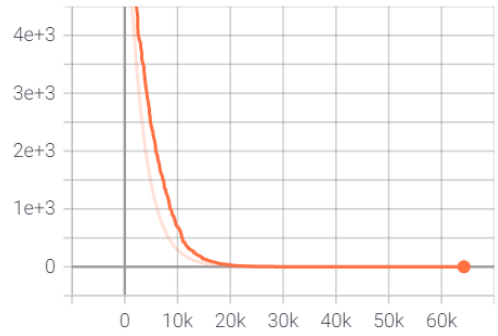

***Figure 4.12**: RetinaNet Total Loss*

Once all the loss data is collated, we can observe the overall trend of the models' losses throughout training. We can observe similar graph attributes that have already been discussed such as the stagnated start in the R-CNN graph (**Figure 4.11**) and its steeper initial decline. The RetinaNet total loss (**Figure 4.12**) mimics its regularisation loss due to the high losses found in **Figure 4.9** so the final loss ends up absurdly high. However, this then results in a consistent line that appears to accurately represent the unsmoothed line, but this is only due to the large zoom out needed. The final total loss values achieved by each network are $2.3x10^{-1}$, $5.4x10^{-3}$, and $2.6x10^{-1}$ showing that the best model for accurate gesture recognition is the Faster R-CNN, followed by the MobileNet, and then the RetinaNet.

4.1.5 Learning Rate

As stated before, learning rate will not be a priority for the finished product but conclusions can still be drawn from evaluating this in our models.

As expected, the smaller and more agile networks (such as the MobileNet and the Faster R-CNN) took far less training time due to their streamlined structure that compensates with lower accuracy. The MobileNet took 7hr 16m 19s, the R-CNN took 2d 1hr 39m 28s and the RetinaNet took 12d 22hr 29m 48s. Do note however, that training time can be affected by other stimuli such as background processes that might have been performed during the training process. With the shorter cycles, this can be avoided but when the larger networks require several weeks of training time, these external effects are more prominent especially since I do not have an additional device that I can dedicate training to (not efficiently anyway). One could propose that the bulkier networks and their long training epochs suffer a serious drawback because of this, and their long processes would be bad for the environment due to taking so much time and power. The RetinaNet's training had to be cut short to only two thirds of its expected steps because of this reason.

4.2 Object Detection Testing

Since a live demo of autonomous object detection and drone control will not be possible for hand-in, the primary evidence of testing will be provided through single image frames with console feedback. Note that processed images undergo filtering, so the outputted videos' colour histograms are skewed producing a blueish tint.

4.2.1 Accuracy

The first attribute of our models that we will evaluate is their accuracy when provided with a small sample from the testing set. These sample images will have similar qualities to the training set, having been taken in the same room under similar conditions. We expect to see bounding boxes around the open palm with high probabilities. Stronger networks such as the RetinaNet would have higher probabilities more consistently while weaker networks such as the MobileNet are expected to have lower estimations, but all estimations should be high assuming training was undergone correctly. Several of these tests were performed and results of interest are listed and explained below.

***Figure 4.13**: MobileNet Accuracy Test (Standard)*

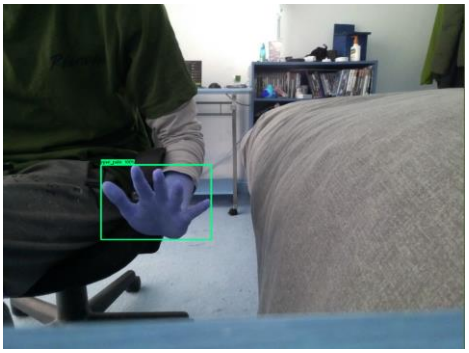

***Figure 4.14**: R-CNN Accuracy Test (Standard)*

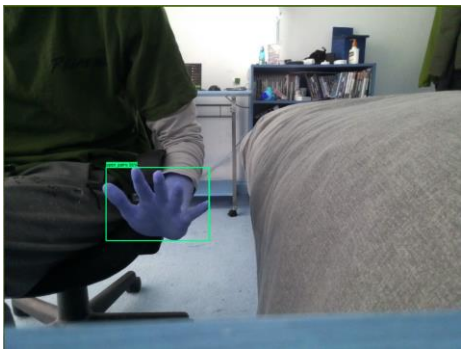

***Figure 4.15**: RetinaNet Accuracy Test (Standard)*

Firstly, we showcase the universal ability of the networks to estimate the presence of an open hand gesture within the image with high probability. Both the MobileNet and R-CNN produced an estimation probability of 100% (**Figure 4.13** and **Figure 4.14**) when provided with an image of an open palm slightly bending forward while the RetinaNet achieved a respectable 95% (**Figure 4.15**). With this, we can ascertain that the models can successfully determine the position and identification of an open palm in select cases. This is an important baseline that the models must achieve to be usable in any scenario and it proves that training was a success.

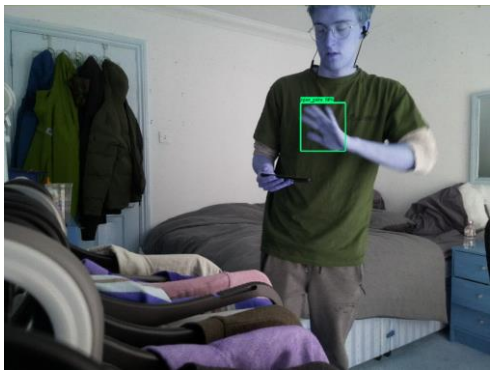

***Figure 4.16**: MobileNet Accuracy Test (Motion Blur)*

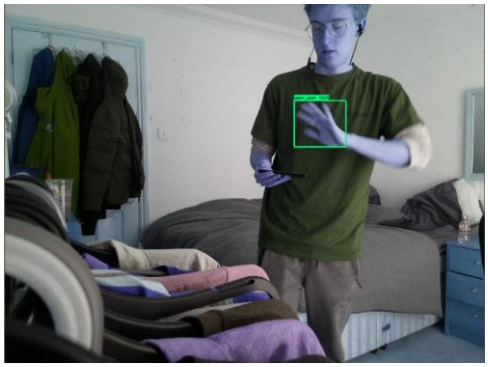

***Figure 4.17**: R-CNN Accuracy Test (Motion Blur)*

***Figure 4.18**: RetinaNet Accuracy Test (Motion Blur)*

This set of testing images depict a typical scenario where an open palm gesture is presented while in motion producing slight motion blur. The first of the three networks, the MobileNet, correctly managed to localise the hand gesture in the image and produced a classification prediction of 88% (**Figure 4.16**) which, although high, lacks in comparison to the R-CNN's interpretation of the image which managed to produce a classification prediction of 100% (**Figure 4.17**). This is expected due to the MobileNet's less complex structure and embedded purpose. Meanwhile the RetinaNet failed to make any prediction of any sort (**Figure 4.18**), whether it be localisation or classification.

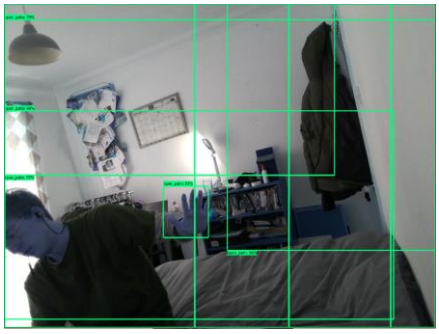

***Figure 4.19**: MobileNet Accuracy Test (Glitch)*

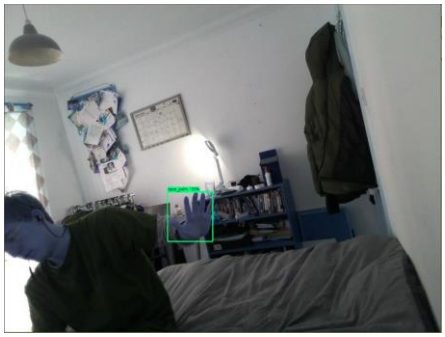

***Figure 4.20**: R-CNN Accuracy Test (Glitch)*

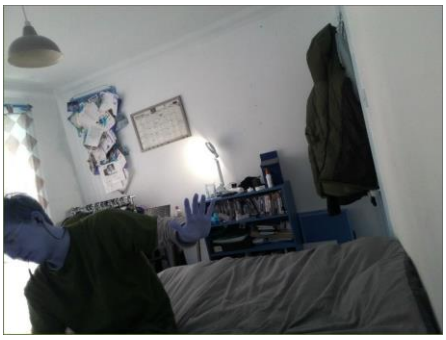

***Figure 4.21**: RetinaNet Accuracy Test (Glitch)*

The third sample from our testing set showcases an unexpected result that was caused by some minor misconfiguring of the configuration file. One of the parameters available describes the maximum number of bounding boxes the model can detect in one input which was mistakenly left to 100. Unfortunately, this caused the MobileNet to produce an absurd number of estimates all over the output image (**Figure 4.19**) which makes inference a lot trickier. Luckily because of how the bounding box output dictionary is produced, we only perform processes using the strongest box data which is usually around the hand. In this case, the MobileNet successfully estimates the location of the hand gesture with a confidence of 64%. As for the other networks, the R-CNN managed to correctly identify the gesture with a final prediction confidence of 100% (**Figure 4.20**) while the RetinaNet failed to produce a bounding box of any sort (**Figure 4.21**). When we extract the bounding box dictionary data however, we do see correct coordinates estimated but the confidence is too low for a visual box to be drawn.

4.2.2 Speed

Speed is paramount in this project since we want the drone to be able to move quickly to follow the detected gesture before it loses track of the hand. The project is already fairly process heavy, so we need as efficient a model as possible. If the model is too slow, then not enough frames will be processed in time and the drone would start moving to follow a gesture that only exists in the past. Although a few seconds of processing might not sound like a lot, when we are processing live video at up to thirty frames per second, it means everything.

To test this quality of the networks, we maintain a counter in the main inference loop that increments every time a frame is extracted from the drone. Since the processing loop contains the code responsible for inferring bounding boxes using the model, we can determine the relative time each network takes to process and display the drone frames. With this, we can then calculate the frames per second (fps)

of the final video which will give us a concrete value to compare between networks.

Firstly, we will evaluate the MobileNet's speed which managed to produce moderately impressive results. This managed to achieve ~3fps which, although low, is enough to produce a comprehensive feed of drone video. This meant that the drone could determine movement controls correctly and adjust its flight controls effectively. Although the drops in prediction accuracy and visual glitches caused some minor drawbacks, this network's light structure made it appropriate to use during fast paced processing.

Our middle ground network structure, the R-CNN, suffered from its larger size and resulted in nearly unintelligible footage at ~0.1fps. This footage is borderline unusable for the drone to move appropriately to follow to the detected gesture. Although the bounding boxes that were produced had higher confidences, this small increase in certainty means almost nothing compared to the drastic drop in processing speed. This network is unsuitable for the purposes of the project but may have some use when used in single image processing. Maybe if the inference algorithm is changed to take a single frame sample every 5 seconds or if a higher performance device were running the model's processes, then maybe this drop in processing speed could be nullified slightly but this is beyond the scope of this project.

The largest of the three, the RetinaNet, ended up performing better than expected, achieving a final fps reading of ~0.1fps. This is similar to its smaller counterpart, the R-CNN, so it suffers the same possibility of not being fast enough at processing concurrent frames where the gesture could completely change positions on screen before the process is complete. However, the lack of prediction success makes it less suitable than the R-CNN as a network for this project.

4.2.3 Adaptability

A machine learning model does not achieve much if it cannot detect specified objects when presented with unseen and familiar data. This section of the report focuses on this issue and tests the models on an image that has been chosen to be as different to the training set as possible. This has been achieved by being in a different room and using the hand of a new subject. Since I am the only subject available, this will be achieved by wearing different clothes. This image is shown below in **Figure 4.22**.

***Figure 4.22**: Adaptability Test Image*

Unfortunately, during testing, none of the networks succeeded in overlaying a bounding box correctly on the image. However, we can still extract their best guess using the bounding box dictionary and the corresponding drone movement. Of the networks, the MobileNet miscalculated the estimate, guessing a close gesture within top left quadrant (probably mistaking the wall lighting for a hand), the R-CNN correctly estimated the hand in the correct position, and the RetinaNet failed to produce an estimate of any sort. Therefore, the most appropriate network for unfamiliar environments is the Faster R-CNN model.

4.3 Autonomous Control Testing

Since no live demonstration of the drone's flying is possible, the autonomous tracking feature will be demonstrated using console outputs when provided with some test images. These images have been chosen to cover most scenarios i.e., hand gestures at a diverse range of distances and angles. Note that in the DJITelloPy library, a positive x velocity causes the drone to travel to the right, a positive y velocity causes the drone to travel upwards and a positive z value causes the drone to travel forwards.

4.3.1 Baseline

***Figure 4.23**: Autonomous Control Test Image (Baseline)*

The first image chosen to test the autonomous drone controls depicts a hand gesture in an ideal position on the screen (**Figure 4.23**). This means the hand gesture is positioned centrally on the screen at an ideal distance. When provided with this bounding box info, we get these raw outputs for the orthogonal velocities:

xSpeed: 15.848677825927734 (≈ 16)

ySpeed: 9.114675545692444 (≈ 9)

zSpeed: -2.953300902238186 (≈ -3)

As expected, the drone is told to travel slightly upwards and to the right (moderately sized xSpeed and ySpeed) so that the bounding box, and consequently the gesture, become more centralised. The depth velocity (zSpeed) on the other hand is a lot smaller and negative so little change is performed on the z axis.

4.3.2 Close Up

***Figure 4.24**: Autonomous Control Test Image (Close Up)*

This next image has been chosen to cover scenarios where the hand gesture is too close to the camera and the drone needs to travel further away to accommodate (**Figure 4.24**). The hand is also slightly off-centre to the right so small values for vertical and horizontal movements are expected. Raw experimental values are listed below:

xSpeed: 16.27837543487549 (≈ 16)

ySpeed: -2.918969678878784 (≈ -3)

zSpeed: -41.874191847548765 (≈ -42)

This supports our estimated drone movement as little change is made to the ySpeed and consequently the vertical height of the drone. Meanwhile, a moderately large positive xSpeed value is calculated

which correlates to moderate movement to the right which is also as expected. As for the depth movement, a large negative value is produced. This causes the drone to move backwards such that the bounding box decreases in size and the gesture is moved back to an appropriate distance. Another successful result.

4.3.3 Far

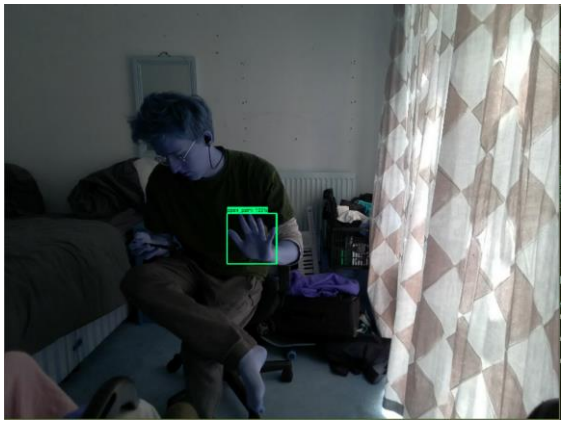

***Figure 4.25**: Autonomous Control Test Image (Far)*

The final test to evaluate the extract the drone velocities uses another typical gesture scenario but this time from relatively far away and in a new quadrant (**Figure 4.25**). Although the hand is more or less centred, the exact coordinates should be in the lower left which would output the remaining unseen behaviour, travelling downwards and to the left while moving closer. The raw experimental output is pasted below:

xSpeed: -6.9772579908370975 (≈ -7)

ySpeed: -6.391579532623291 (≈ -6)

zSpeed: 21.65681590272508 (≈ 22)

These values support our estimates since moderately sized negative values are produced for the vertical and horizontal axes (which will cause the drone to fly downwards and to the left) and a large positive value for the depth speed (which will cause the drone to fly forwards).

5 Conclusion

In total, three networks were configured and trained to detect and classify an open palmed gesture within live footage captured from a drone. Several TensorFlow architectures were chosen to diversify structure topologies such that meaningful conclusions can be made. Each network has been evaluated using a variety of parameters that assess their performance under several scenarios including, but not limited to, their adaptability, accuracy, and speed.

The most effective model when it came down to pure raw loss values ended up being the R-CNN architecture due to its higher resolution processing when compared to the streamlined MobileNet. This goes against our predictions as the RetinaNet had a higher listed COCO mAP score but the time needed to train this network ended up being too large, so training had to be cut short. Even with complete training however, the trends of the graphs showed that the RetinaNet's loss would fail to improve better than the other networks.

When presented with testing images, the R-CNN proved to be the most accurate and adaptable achieving 100% predictions in most cases. During experimental tests however, the slow processing of the R-CNN network resulted in unsatisfactory speeds, so the slightly underachieving MobileNet is deemed the most suitable instead as its accuracy was still satisfactory while having the highest processing speed.

When it came to controlling the drone autonomously, a preconfigured python library [43] was used that allowed the easy transmission of velocity values that are calculated using the predicted bounding box attributes. When tested using console outputs, correct values were produced and when tested during physical experiments, the drone flew slowly and appropriately, accounting for model processing delays.

As a whole, I would say that the original objectives stated in the Introduction have been achieved to good effect. Live drone video has been successfully extracted and processed to label bounding box predictions for an open palm gesture. The coordinates of these bounding boxes are then accessed and used to calculate necessary orthogonal velocities to keep the gesture centred. In future works, I think the project could be improved by training an even faster network architecture and using a more diverse collection of training data to improve accuracy. I would also extend the list of possible gestures to give the user greater control over the drones positioning, such as a 'move closer' gesture or even a panning mode. It would have also been nice to finish the autonomous rotation code but, even without it, the final implementation still does justice to my original vision.

Appendix A

***Table 2**: TensorFlow Model Zoo*

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes

EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes

Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks
ExtremeNet (deprecated)	--	--	Boxes
ExtremeNet	--	--	Boxes

Bibliography

[1] Nanonets, “Scope of use across industries,” [Online]. Available: https://nanonets.com/drone/.

[2] JannaHuuskonenTimoOksanen, “Soil sampling with drones and augmented reality in precision agriculture,” *Computers and Electronics in Agriculture,* vol. 154, pp. 25-35, 2018.

[3] A. N. &. L. R. Vikram Puri, “Agriculture drones: A modern breakthrough in precision agriculture,” *Journal of Statistics and Management Systems,* vol. 20, no. 4, pp. 507-518, 2017.

[4] M. A. N. A. S. A. Hesham Ismail, “Enhance PV Panel Detection Using Drone Equipped With RTK,” in *ASME 2020 International Mechanical Engineering Congress and Exposition*, 2021.

[5] O. Moolan-Feroze, K. Karachalios, D. N. Nikolaidis and A. Calway, “Improving drone localisation around wind turbines using monocular model-based tracking,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, 2019.

[6] B. E. Schäfer, D. Picchi, T. Engelhardt and D. Abel, “Multicopter unmanned aerial vehicle for automated inspection of wind turbines,” in *Mediterranean Conference on Control and Automation (MED)*, Athens, 2016.

[7] B. A. A. M. T. J. B. Ashim Khadkaa, “Non-contact vibration monitoring of rotating wind turbines using a semi-autonomous UAV,” *Mechanical Systems and Signal Processing,* vol. 138, 2020.

[8] J. Seo, J. P. Wacker and L. Duque, “Evaluating the use of drones for timber bridge inspection,” General Technical Report (GTR), U.S. Forest Service, Wisconsin, 2018.

[9] R. Obradović, I. Vasiljević, D. Kovačević, Z. Marinković and R. Farkas, “Drone Aided Inspection during Bridge Construction,” in *Zooming Innovation in Consumer Electronics International Conference (ZINC)*, Novi Sad, 2019.

[10] A. R. P. J. B. L. a. G. C. Bo Shang, “Vision-based non-GPS UAV guidance for bridge inspection,” Missouri, 2020.

[11] J. Baghirov, “The use of drones in oil and gas logistics,” Molde University College, Molde, 2018.

[12] P. Gajalakshmi, J. V. Satyanarayana, G. V. Reddy and S. Dhavale, “Detection of Strategic Targets of Interest in Satellite Images using YOLO,” in *International Conference on Computer, Communication and Signal Processing (ICCCSP)*, Chennai, India, 2020.

[13] E. Páli, K. Máthé, L. Tamás and L. Buşoniu, “Railway track following with the AR.Drone using vanishing point detection,” in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, Cluj-Napoca, Romania, 2014.

[14] F. Flammini, C. Pragliola and G. Smarra, “Railway infrastructure monitoring by drones,” in *2016 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC)*, Toulouse, France, 2016.

[15] A. S. A. A. D. S. Arun Kumar Singh, “Vision based rail track extraction and monitoring through drone imagery,” *ICT Express,* vol. 5, no. 4, pp. 250-255, 2017.

[16] G. W. Z. L. J.-N. H. Haotian Zhang, “Eye in the Sky: Drone-Based Object Tracking and 3D Localization,” in *MM '19: Proceedings of the 27th ACM International Conference on Multimedia*, New York, 2019.

[17] A. Sehrawat, T. A. Choudhury and G. Raj, “Surveillance drone for disaster management and military security,” in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida, India, 2017.

[18] M. A. I. F. A. N. Naveed Anwar, “Construction Monitoring and Reporting using Drones and Unmanned Aerial,” in *The Tenth International Conference on Construction in the 21st Century (CITC-10)*, Colombo, Sri Lanka, 2018.

[19] D. G. P. N. V. M. Balmukund Mishra, “Drone-surveillance for search and rescue in natural disaster,” *Computer Communications,* vol. 156, pp. 1-10, 2020.

[20] L. L. P. W. W. Sven Mayer, “Drones for Search and Rescue,” in *1st International Workshop on Human-Drone Interaction*, Glasgow, 2019.

[21] M. M. J. F. Pompílio ARAÚJO JR., “Towards Autonomous Investigation of Crime Scene,” *Sensors & Transducers,* vol. 234, no. 6, pp. 30-36, 2019.

[22] S. S. E. B. M. S. G. N. K. Osman Tarik Cetinkaya, “A Fuzzy Rule Based Visual Criminal Tracking,” in *2019 4th International Conference on Computer Science and Engineering (UBMK)*, Samsun, Turkey, 2019.

[23] M. W. A. S. G. H. Evangeline Corcoran, “Automated detection of wildlife using drones: Synthesis, opportunities and constraints,” *Methods with Ecology and Evolution,* vol. 12, no. 6, pp. 1103-1114, 2021.

[24] M. M.-P. Jesús Jiménez López, “Drones for Conservation in Protected Areas: Present and Future,” *Drones,* vol. 1, no. 3, p. 10, 2019.

[25] R. M. S. M. B. T. T. P. S. W. A. D. K. R. R. S. I. R. A. T. L. P. K. Jarrod C. Hodgson, “Drones count wildlife more accurately and precisely than humans,” *Methods in Ecology and Evolution,* vol. 9, no. 5, pp. 1160-1167, 2018.

[26] A. P. C. R. A. G. S. M. K. N. A. L. J. M. C. R. P. G. B. S. J. P. T. A. J. W. J. E. W. Paul A. Butcher, “The Drone Revolution of Shark Science: A Review,” *Drones,* vol. 5, no. 1, p. 8, 2021.

[27] T. P. P. A. P. C. B. H. V. M. P. L. B. a. B. R. C. Paul A. Butcher, “Beach safety: can drones provide a platform for sighting sharks?,” *Wildlife Research,* vol. 46, no. 8, pp. 701-712, 2019.

[28] P. R. M. J. V.-M. O. M. H. A. P. S. W. S. L. Claire Burke, “Requirements and Limitations of Thermal Drones for Effective Search and Rescue in Marine and Coastal Areas,” *Drones,* vol. 4, no. 3, p. 78, 2019.

[29] L. N. Hung and L. S. Bon, “A quadcopter-based auto cameraman system,” in *2016 IEEE Virtual Conference on Applications of Commercial Sensors (VCACS)*, Piscataway, New Jersey, 2017.

[30] C. Huang, Z. Yang, Y. Kong, P. Chen, X. Yang and K.-T. T. Cheng, “Learning to Capture a Film-Look Video with a Camera Drone,” in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, 2019.

[31] C. A. L. C. N. P. C. a. A. S. E. Seth Kielbasa, “Otto: The Personal Cameraman,” 2015. [Online]. Available: http://www.ecs.umass.edu/ece/sdp/sdp15/team07/docs/Otto_FPR_Report.pdf.

[32] geaxgx1, “tello-openpose,” [Online]. Available: https://github.com/geaxgx/tello-openpose.

[33] hanyazou, “TelloPy,” [Online]. Available: https://github.com/hanyazou/TelloPy.

[34] C. P. C. Lab, “OpenPose,” [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose.

[35] Ubotica, “Tellocv Tracker,” [Online]. Available: https://github.com/Ubotica/telloCV/.

[36] N. V. Francisca Vasconcelos, *Person-following UAVs,* San Diego: University of San Diego, 2016.

[37] A. Sarkar, K. A. Patel, R. G. Ram and G. K. Capoor, “Gesture control of drone using a motion controller,” in *International Conference on Computer Systems and Industrial Informatics (ICCSII)*, Sharjah, United Arab Emirates, 2016.

[38] J. W. Bin Hu, “Deep Learning Based Hand Gesture Recognition and UAV Flight Controls,” *International Journal of Automation and Computing,* vol. 17, pp. 17-29, 2020.

[39] K. Natarajan, T.-H. D. Nguyen and M. Mete, “Hand Gesture Controlled Drones: An Open Source Library,” in *1st International Conference on Data Intelligence and Security (ICDIS)*, South Padre Island, TX, USA, 2018.

[40] C. C. Aggarwal, *Neural Networks and Deep Learning,* New York: Springer International, 2018.

[41] “TensorFlow,” [Online]. Available: https://github.com/tensorflow.

[42] "DJI Tello Ryze," [Online]. Available: https://www.ryzerobotics.com/tello.

[43] damiafuentes, "DJITelloPy," [Online]. Available: https://github.com/damiafuentes/DJITelloPy.

[44] "OpenCV," [Online]. Available: https://opencv.org/.

[45] tzutalin, "LabelImg," 21 July 2017. [Online]. Available: https://pypi.org/project/labelImg/1.4.0/.

[46] T. Gilbert, "Tensorflow Object Detection with Tensorflow 2: Creating a custom model," 27 July 2020. [Online]. Available: https://gilberttanner.com/blog/tensorflow-object-detection-with-tensorflow-2-creating-a-custom-model.

[47] T. Gilbert, "XML to CSV conversion," 30 November 2020. [Online]. Available: https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model/blob/master/xml_to_csv.py.

[48] T. Gilbert, "TFRecord conversion," 30 November 2020. [Online]. Available: https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model/blob/master/generate_tfrecord.py.

[49] O. Elisha, "Real-time Object Detection using SSD MobileNet V2 on Video Streams," 17 September 2020. [Online]. Available: https://heartbeat.fritz.ai/real-time-object-detection-using-ssd-mobilenet-v2-on-video-streams-3bfc1577399c.

[50] A. Pujara, "Image Classification With MobileNet," 4 July 2020. [Online]. Available: https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470.

[51] Z. Deng, "Multi-scale object detection in remote sensing imagery with convolutional neural networks," *ISPRS Journal of Photogrammetry and Remote Sensing,* p. 145, 3 April 2018.

[52] S.-H. Tsang, "Review: RetinaNet — Focal Loss (Object Detection)," 24 January 2019. [Online]. Available: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4.