UAV Obstacle Avoidance by Applying Deep Learning

by

Wenyu Zhu

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
Aug 7, 2021

Keywords: UAV, Obstacle Avoidance, Deep Learning, Computer Vision

Approved by

Richard Chapman, Chair, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering
Bo Liu, Assistant Professor Computer Science and Software Engineering
Shubhra Karmaker, Assistant Professor Computer Science and Software Engineering

Abstract

In the last decades, many algorithms in the artificial intelligence field, including machine learning, deep learning and so on, have been implemented for not only computer vision, but also in mobile vehicles. Thus, to acquire a fast responding and accurate algorithm, this project presents several tests of different deep learning models included in an obstacle avoidance algorithm. In this experiment, a Mavic Air UAV is used for testing this algorithm. The system is designed as two separate parts. One is a pre-operation machine learning system (Keras machine learning framework), the other one is a ground-station control system (a UWP application). CNN, Time-series, semantic segmentation and instance segmentation models are trained and tested with custom dataset in this project. The test flight is taken at a parking lot where the main obstacles are the trees and the light pole in the median of the parking lot. The model training results show the multi-stage models outperforms the End To End (E2E) models according to various metrics. The test flight shows that even though the model has a acceptable performance, the long inference time of the large size of the model limits the flying speed too much to prevent collisions.

Acknowledgments

During the experiment and writing of the dissertation, I was received a lot of help.

I would first thank my advisor, Professor Richard Chapman, who supported me to realize my ideas, helped me with the equipment, and evaluating my proposed ideas. Your feedback enlightened me and led me on the right path in the research.

During my Ph.D. years in Auburn, the Ruggiero family have given me heartwarming hosting, made me feel like a part of the family ,and your enthusiasm and taking care of me held me strong when I was alone in this foreign country. I appreciate you for everything you have done for me spiritually, mentally ,or materially.

In addition, I would like to thank my friends and my family for their counsel. Even though it has been a difficult period in the last year for everybody, you all have been there for me with your sympathy and support. Finally, I couldn't go through the tough years without my parents' guidance.

Table of Contents

Abstract ii

Acknowledgments iii

1 Introduction 1

2 Literature Review 4

3 Devices 11

3.1 UAV 11

3.2 Deep Learning System 12

3.3 Source of Images 13

3.4 Application 15

3.5 Model Conversion 16

4 Deep Learning Models 19

4.1 CNN 19

4.2 Time Series Model 21

4.2.1 RNN Models 21

4.2.2 A Proposed Simple RNN Model for Obstacle Avoidance 23

4.3 Data Augmentation 25

4.3.1 Traditional Data Augmentation method 25

4.3.2 GANs Data Augmentation method 26

4.3.3 Data Augmentation with GauGANs 27

4.4 Semantic Segmentation and Instance Segmentation 29
4.4.1 Computer Vision Tasks 30
4.4.2 YOLO 31
4.4.3 DeepLab 34
4.4.4 R-CNN 34
4.4.5 Fast R-CNN 35
4.4.6 Faster R-CNN 36
4.4.7 Mask R-CNN 38
4.4.8 Two-Stages UAV Obstacle Avoidance Deep Learning model 39

5 Experiment Design 44

6 Experiment Result 51
6.1 Application Development 51
6.2 ResNet Family Models Development Result 52
6.2.1 ResNet152 52
6.2.2 ResNet101 52
6.2.3 ResNet50 55
6.3 RNN Family Models Development Result 56
6.3.1 ResNet152 based GRU Model 56
6.3.2 ResNet101 based GRU Model 57
6.3.3 ResNet50 based GRU Model 57
6.4 ResNet Family Models Development Result Using GANs Data Augmentation . 58
6.4.1 ResNet152 Using Synthetic Images Data Augmentation 59
6.4.2 ResNet101 Using Synthetic Images Data Augmentation 59
6.4.3 ResNet50 Using Synthetic Images Data Augmentation 61
6.5 Semantic or Instance Segmentation based Multi-Stage Model 63

6.5.1 Semantic Segmentation based Multi-Stage Model 63
6.5.2 Instance Segmentation based Multi-Stage Model 63
6.6 Test Flight 65

7 Conclusion 68

8 Future Work 69

References 70

List of Figures

3.1 DJI Mavic Air 12
3.2 Same Object in Different Categories 14
3.3 Set Waypoints Page 16
3.4 FPV Page 17
3.5 ONNX Compatibility [23] 18
4.1 Residual Block [26] 21
4.2 ResNetV1 vs ResNetV2 [23] 22
4.3 LSTM Recurrent Cell [70] 23
4.4 GRU Recurrent Cell [31] 24
4.5 Proposed RNN Model 25
4.6 Source Image 29
4.7 GauGAN Synthetic Image 30
4.8 YOLO Model [52] 32
4.9 DeepLabv2 Model [7] 35
4.10 Atrous convolution [6] 36
4.11 Atrous Sptial Pyramid Pooling [6] 37
4.12 R-CNN Model [20] 37
4.13 Fast R-CNN Model [21] 38
4.14 RPN 39
4.15 AnchorBoxes 40
4.16 IoU 41

4.17 Faster R-CNN Model [55] 42
4.18 Mask R-CNN Model [25] 43
4.19 Proposed Semantic Segmentation based Model 43
4.20 Proposed Instance Segmentation based Model 43
5.1 System Flow Chart 45
5.2 Airspace Profile 46
5.3 Path Planning Mechanism 47
5.4 Waypoint Pruning Mechanism 49
5.5 560 Devall Dr., Auburn, Alabama 49
5.6 The Obstacles in the Field 50
6.1 Model Training Plot of ResNet152 53
6.2 Model Training Plot of ResNet101 54
6.3 Model Training Plot of ResNet50 55
6.4 Model Training Plot of ResNet152 based GRU Model 56
6.5 Model Training Plot of ResNet101 based GRU Model 57
6.6 Model Training Plot of ResNet50 based GRU Model 58
6.7 Model Training Plot of ResNet152 Using Synthetic Images 59
6.8 Model Training Plot of ResNet101 Using Synthetic Images 60
6.9 Model Training Plot of ResNet50 Using Synthetic Images 61
6.10 Model Training Plot of Semantic Segmentation based Model 64
6.11 Model Training Plot of Instance Segmentation based Model 66

List of Tables

6.1 E2E Models Accuracy (Training Accuracy/Validation Accuracy) 62

6.2 E2E Models Evaluation 62

6.3 Multi-stage Models Accuracy 65

6.4 Multi-stages Models Evaluation 65

Chapter 1

Introduction

The idea of the delivery drone is arose in this decade because it's convenient and labor-saving. To make the drone arrive at a destination and return home safely and quickly, an excellent obstacle avoidance algorithm is necessary. Obstacle avoidance for Unmanned Aerial Vehicles (UAVs) has been developed for many years; however, most of the algorithms are low-level intelligence, for example, the stereo-vision-based algorithm. In the last decades, many algorithms in the artificial intelligence field, including machine learning, deep learning and so on, have been implemented in not only computer vision, but also mobile vehicles. Because of the advantage of deep learning in image processing, it is showing excellent results for solving a wide variety of robotic tasks in object recognition, obstacle avoidance, and control [4]. However, there are only a few algorithms are focusing on processing color images, which contain vast amount of information. Thus, to acquire a fast-responding and accurate algorithm, this project presents several tests of different deep learning models included in an obstacle avoidance algorithm. The purpose of this project is to compare several different deep learning models by developing a Universal Windows Platform (UWP) application with a deep learning model that can navigate the UAV from the start waypoint to the final waypoint without collision.

To use the deep learning technique to navigate drones, a deep learning model can be regarded as a brain that processes the images captured from the 'eye' of a UAV, a monocular camera, then determines how to avoid the obstacles. A preliminary requirement of a deep learning model is having the ability to do the same determination in an obstacle avoidance algorithm as a drone pilot does. If people train the models with the dataset that has the same

training distribution as a pilot does, the obstacle avoidance behavior can be similar to the pilot's behavior; moreover, the more data there is in the training dataset, the more similar the behaviors are. Thus, using a model with an image input to avoid obstacles on a UAV seems plausible.

End To End (E2E) learning is a hot topic in the Deep Learning field for taking advantage of Deep Neural Network's structure, composed of several layers, to solve complex problems. Similar to the human brain, each neural network's layer can specialize to perform intermediate tasks necessary for such problems. [22] The model to be tested at first in this project is an E2E models for object recognition. The main advantage of end-to-end models is that they can figure out the relations between inputs and output without the interference of humans, so maybe the model doesn't navigate the UAV according to the location of objects but something different to how humans do it. There are also many limitations to E2E models. A huge amount of data is necessary because the incorporation of some prior knowledge into the training is considered a key element that will allow an increase in performance in many applications. For E2E learning not integrating this prior knowledge, more training examples must be provided.

The info in a colored image is enough to tell the objects' types [68]. This has been proven by many models, and the number is still increasing. However, it is still a myth that a single colored image has enough information to tell how to avoid an obstacle. In this paper, several models are trained and tested with a dataset that is collected during a pilot-controlled flight. After exploring E2E CNN models, (such as Residual Neural Network (ResNet) family models), and finding that they do not work well, several auxiliary methods to enrich the information hidden in the images are implemented, such as inputting both the semantic or instance mask of the original image and the colored image into the model. After applying Generative Adversarial Networks (GANs), Recurrent Neural Network (RNN), and semantic and instance segmentation, the semantic and instance segmentation wins the competition. In the proposed model that contains a semantic and instance segmentation model, there are multiple stages. The model accuracy outperforms the others and the test flight also shows that the algorithm works fine. One of the reasons is using multi-stage models requires less data, and a pretrained model in one of the stages contributes a lot in reducing the required data.

The system is designed as two separate parts. One is tagged as a pre-operation machine learning system (Keras machine learning framework), and the other one is tagged as a ground-station control system (a UWP application). The pre-operation machine learning system trains the models to check the validation accuracy, which is the metric of the model. The UWP application controls the UAV during the test flight. It holds the model and sends the output of the model to the drone.

The result shows that the E2E models have difficulty figuring out the relation between labels and the input itself, because the overfitting problem persists in the models. On contrary, the multi-stages models have better performance overfitting problems. The validation accuracy reaches over 80% and there is no fluctuation in the validation accuracy and loss. The test flight show that the multi-stage models have a long inference time, 2.719s for semantic segmentation based model and 1.46s for instance segmentation based model.This shows that the model is suitable for UAV obstacle avoidance when the flying speed is relatively slow; otherwise, there is a safety concern.

Chapter 2

Literature Review

CNN (Convolutional Neural network) is used with a lot of indoor obstacle avoidance examples to show the effectiveness of making decisions in UAV obstacle avoidance [59]. It takes raw multi-channel images as input and produces the commands for the control system as output. Also, it builds a CNN network for recognition and a fully connected network for decision making. In a CNN network, vanilla 2D convolution is used. Rectified Linear Unit (ReLU) is chosen to be the non-linear activation function, and regular pooling and stride are also involved. For the fully connected layer, a designer normally puts several of neurons at the end and outputs the highest value (confidence), in the vector that represents the results of each neuron; however, this paper suggested a different decision-making mechanism. Instead of choosing the command with the highest confidence, it computes the inner product of discretized angular velocity and the confidence of them. Thus, the command doesn't have to be one of the preset velocities, such as turn left or turn right. It can be any value in the range of angular velocity. The advantage of this technique is that it yields the confidence of each angular velocity class, which means it can be reasonable even if the confidences are close; however, this model was only tested in an indoor environment and on a four-wheel robot. This technique doesn't fit on UAVs because outdoor UAVs do not have to have an extremely precise avoidance direction. Outdoor space is big enough to do the discrete-classified command instead of a complicated avoidance environment.

Another algorithm involves using a single forward-facing camera on a UAV training a CNN for calculating the depth from an image [5]. The matrix with depth information is then sent to a control algorithm that drives a quadrotor away from obstacles. It has a CNN based on

the neural network in [16] and uses scale-invariant error in log-space as its loss function. All the calculation in that project is done at a local laptop graphic processing unit (GPU), so the drone must be connected to the laptop. As a result, the drone was able to avoid indoor obstacles, like tables and whiteboards, and show wonderful depth images on the screen. Although this project has excellent results for avoiding indoor obstacles, it's still a distance-based obstacle avoidance algorithm, which costs more time on the path planning stage.

A novel and memory-efficient deep network architecture named UAVNet for small UAV to achieve obstacle detection in the urban environment is proposed in [8]. In contrast to the state-of-art architecture, UAVNet has fewer parameters but acceptable accuracy and is suitable for UAV-type applications. A core structure is introduced in the UAVNet architecture, which is depthwise convolution. A standard convolution applies a 3x3 filter first, followed by a batch normalization and finally the ReLU nonlinearity. In UAVNet, the standard convolution is replaced by first a 3x3 depthwise convolution and followed by a 1x1 point-wise convolution. After each operation, a ReLU activation is inserted. For the depthwise convolution used in UAVNet, a single filter is applied to each input feature map rather than the standard convolution in which the filters do the convolution operation and sum over all the previous feature maps to generate the output. After the depthwise convolution, a 1x1 projection convolution is used to combine the output of depthwise convolution. UAVNet has only 2.23M parameters and 141 M floating point operation per second (FLOPs), which is memory efficient, in order to deploy on the quadrotor UAV. The experiments results show that UAVNet can detect obstacles up to 15 fps, which is good enough for real-time applications. On the other hand, the accuracy of UAVNet is only 80% with the ImageNet-2012 dataset.

The You Only Look Once (YOLO) algorithm was developed in 2016. It has been evaluated as a powerful object detection algorithm base on deep learning. A method to perform real-time object detection on-board a UAV using the YOLOv2 algorithm was demonstrated [60]. This paper highlights the comparison between a traditional method (Aggregated Channel Features (ACF)) and YOLO, and the result shows that YOLOv2 outperforms ACF with an average accuracy of 85.9% vs. 62.4% at the same frame rate. Also, the computation speed of a detector is pivotal to UAV flight. The ACF detector reaches a mean frame rate of 0.54 fps while the

YOLO detectors reach a frame rate of 4.53 fps. In this paper, the key purpose is to provide invaluable visual information during incidents and help rescue workers and coordinators better tackle the situation at hand, thus, the job of YOLOv2 is to help decision-makers assess the situation more efficiently and coordinate the operation with greater ease than would normally be the case. Since this is an object detection oriented system, it's different from our method.

There is another YOLO algorithm application on UAV in [36]. Object detection with a single-shot image on a constrained resource, such as a UAV, is developed, and the trade-offs of using the YOLO algorithm are discussed. For faster and more efficient computation on UAV, a smaller YOLO model, the Tiny-YOLO model is selected to train and recognize the object. Instead of using the vanilla Tiny-YOLO model, they adapted it to detect only one class by changing the number of filters, layers, etc. There are 4 different models, SmallYoloV3, TinyYoloVoc, TinyYoloNet, and DroNet, which are created for competing with each other. DroNet beats the others because it is capable of 5 to 18 FPS on different platforms while achieving 95% detection accuracy. However, because the objects in the project are limited to the top view of cars, it is clear it cannot be applied in obstacle avoidance design.

A research implementing the Siamese Network on a UAV for real-time object-to-features vectorization into a Euclidean metric space is introduced in [19]. This approach is significantly faster due to a smaller number of operations required and is also sped up using certain optimization techniques. Siamese-type network architecture is deployed for learning face representation that is computed in a form of 128-dimensional descriptor vector. The input is a pair of face images that either belong to a single person or two different people. Two networks that share the same parameters then compute a descriptor vector for both items in a pair. This neural network has multiple subnetworks, each of them is a neural network with three convolutional-pooling layer pairings and two fully connected neuron layers. In every case, the last layer of the network consists of 128 neurons, which means the output of each subnetwork is a 128-sized vector. At last, using squared normalization to compare outputs from two twins, the system knows what the object is. This network computes its single object forward-pass in about 13% of required operations in FaceNet. Thus it brings object vectorization closer to being real-time in embedded systems. This method is hard to apply to outdoor UAVs because the objects in outdoor

scenes usually number more than one. The complexity of learning the differences between multiple types of objects would be far greater than learning only different faces.

The input of a deep learning neural network doesn't have to be just one single set of data but also can be multiple vectors of data, like the network in [35]. In this paper, a CNN network based on Alexnet with two inputs is used for UAV navigation. One is the normal RGB images and the other one is the depth map from sensors. Because the number of channels of RGB image and depth image is 3 and 1 respectively, the channel in-depth image must be adjusted to 3 by replicating the original channel to make them symmetric. Then depth and normal images are convoluted at two different networks with the first three layers and are concatenated at the fourth layer. Thus, the final output of this CNN uses both information of two inputs. Since this project used the NYUv2 RGB-D dataset, which contains many diverse indoor scenes, and provides 3D point clouds for each image, it doesn't fit on the outdoor scene based deep learning model training. The main difference between the techniques in the paper above and this paper is that the former has not only an RGB image also a distance information included data image; however, the later has only an RGB image.

Furthermore, in paper [69], there is another model that contains two inputs. The probabilistic CNN (pCNN) proposed in the paper has two separate input branches to take both an RGB image and its corresponding sparse depth map as inputs. Various methods can be utilized for sparse depth map generation. The feature tracking based approach used in ORB-SLAM [43] is leveraged in this paper, due to its low computational complexity, high accuracy, and robustness. Each branch processes the respective input using a convolutional layer, followed by a max-pooling layer. The outputs of the two branches are concatenated together and processed by another convolutional layer. After that, 8 successive residual modules [26] of ResNet18 are applied to perform further feature extraction. The obtained feature maps are then resized by a set of de-convolutional layers. To capture the fine details of an image, skip connections are applied in the network to form a 'U-NET' structure. Finally, the feature maps are processed by two separate branches, each of which consists of a convolutional layer to generate an individual output, a depth map and a confidence map. After having the two outputs, a path planning system including, Ego Dynamic Space (EDS) [40], waypoints selection and control stages. The

effective distances in EDS indicate whether the UAV is going to collide on the object, i.e. a cluster of the pixels which has effective distances larger than 0 is identified as an open area. As a result, this system has a high accuracy on model training, simulation experiments, and real indoor experiments. It is worth noting the real indoor obstacle avoidance performance has a 100% success rate, even when the training data is completely different from the experiment environment. The technique above is no doubt outstanding; however, it is incompatible with the outdoor UAV obstacle avoidance. The inputs need not only RGB image but also a corresponding depth map, which requires computation time spent on generating a sparse depth map. Also, on path planning stages, the effective distance of every pixel must be calculated to form a traversable blob, which also spends unnecessary time.

A model with multiple inputs normally is implemented when sufficient features are not extracted from a single type of input, and the features from the inputs can all contribute to the model learning. As shown in the paper [32], an RGB-D residual encoder-decoder architecture, named RedNet, is proposed to generate an indoor semantic segmentation. Since the job is to generate a semantic mask for an image, using only the RGB colored image info can be insufficient, so the model adopts an architecture that has two inputs. One of them is an RGB-colored image, and the other one is the depth map of that image. The two branches of the inputs have the same structure as the ResNet50 in [26], except for the last two layers, the global average pooling layer, and the fully connected layer. At the end of the encoder, the features of the two branches are fused by element-wise summation. In both the encoder and decoder parts, the residual block is implemented to ensure the info doesn't leak during the training. Since the model is used for image generation, the metric adopted in the paper is the mean intersection-over-union (mIoU) score. The proposed model outperforms most of the state-of-the-art methods on the SUN RGB-D testing set [57] by reaching 47.8% mIoU. This model with multiple inputs shows a clever technique to feed an auxiliary input into the system and have an outstanding performance.

As an example of applying semantic segmentation on obstacle avoidance algorithm, paper [42] combines the semantic segmentation model ICNet [71] and traditional image processing. Once the semantic image is generated, the following steps are implemented: setting 1 to

the traversable area and 0 to the others, extracting points that may correspond to boundaries of the traversable area, applying clustering to the extracted points based on the nearest neighbor scheme, computing a circumscribed rectangle surrounding the largest cluster, and drawing a straight line crossing the two midpoints corresponding to the two short sides of the rectangle. After the geometry calculation on the segmentation map, a target point on the image can drive the robot and avoid the obstacles on the road. This technique performs excellently, but it is adopted to terrain vehicles, not aerial vehicles. The idea of setting a certain part of labels as traversable space for UAV is worthy to try.

The idea of using multiple inputs is also reflected in the paper [27]. The model in this paper borrows the idea of RedNet [32] and is implemented to continuously generate semantic images when the robot is running. However, different from the RedNet, this model implements a two-stage RGB-D semantic segmentation network architecture, which generates a binary mask in which 0 represents the background, and 1 represents the traversable road. After generating a semantic image using RedNet at stage one, a mask can be drawn from the semantic image by setting the pixel inside of road contour 1 and outside 0. Now there are three categories on the image: road, obstacle on the road, and others. Objects within road contours are all mapped to obstacle class during training. At stage two the RedNet is applied again. The two inputs are the mask filtered original colored image and the mask filtered depth map. With the binary map of the traversable road, the obstacle can be easily avoided by applying morphological processing on it. The proposed method outperforms other semantic segmentation architectures both in indoor and outdoor scenarios.

One example of applying instance segmentation to a UAV's obstacle avoidance system is proposed in [37]. This paper focuses on validating the UAV flying at a low altitude in an agricultural environment by developing an obstacle avoidance algorithm. Normally, to have excellent performance on obstacle avoidance, active sensors are required; however, for small UAVs, fancy sensors are financially and dimensionally expensive. Thus, for those UAVs with monocular vision which does not provide depth information, a machine learning model, Faster Region-based Convolutional Neural Network (Faster R-CNN), was trained for tree trunk detection. In this paper, the dataset is collected by manually flying a UAV and recording a video

at the same time. The models selected as the base of Faster R-CNN are the pre-trained Inception V2 model, ResNet50 model, and Scratch Models. The result shows that pre-trained model Inception V2 based Faster R-CNN has the best result on average precision. This paper develops the excellent idea that using instance segmentation in monocular UAV's obstacle avoidance system is a great choice to save the cost on sensors and still maintain a good detection accuracy.

Chapter 3

Devices

3.1 UAV

Initially, a decision of which UAV should be used in this project must be made, and the selected UAV has to be supported by a ground station SDK. Parrot SA is a French wireless UAV products manufacturer company, their products are quite famous around the world. The company not only has products for individuals but also is cooperating with the military in different countries. Their products, Bebop, Disco, Bebop 2 and Anafi, are supported by a ground station SDK, Olympe, with functions of image processing, making commands, and UAV state feedback, which makes it suitable for this project [49].

Another investigated UAV production company is DJI. DJI is a Chinese technology company, a world-leading manufacturer of commercial unmanned aerial vehicles (commonly known as ”drones”) for aerial photography and videography [63]. DJI drone products are various, but those who are supported by a ground station SDK are some in Matrice series, Phantom series, Mavic series, and Manifold series. Since the main task of the UAV in this project is testing the obstacle avoidance algorithm, there is no need to buy an expensive Matrice, Phantom or Manifold UAV. Also, the only available laptop is equipped with a Windows system, a Windows SDK compatibility is highly considered for the product selection. Thus, a cheap, Windows SDK supported Mavic Air is the winner among all candidate DJI products.

Given that I have more experience using DJI is more than using Parrot, in this experiment, a Mavic Air UAV (in figure 3.1) is used for testing this algorithm. Mavic Air is a product from DJI company and is equipped with parts such as a 3-axis gimbal camera, which can shoot 4K

100 Mbps video, magnesium alloy brackets, which can support the seven onboard cameras, seven auxiliary cameras to measure the distance to an obstacle in the forward, backward and downward direction. Also, since the test is conducted in an outdoor environment, a satellite navigation system is needed for waypoint assignment. Mavic Air has both GPS and GLONASS onboard, and this makes it suited for this experiment [14].

Figure 3.1: DJI Mavic Air

3.2 Deep Learning System

For system configuration, there are a lot of images taken by a real UAV that serves as either of training datasets or testing datasets. This system runs on a local Linux PC, which has a large memory and a powerful GPU processor, so there is no need to worry that the pre-operation system is incapable of learning a sophisticated model.

Currently, there are several AI Frameworks that are widely used, such as Pytorch, CNTK, Tensorflow, etc. Pytorch was open-sourced by Facebook in January 2017, and it offers dynamic computation graphs, which let users process variable-length inputs and outputs, which is useful when working with RNNs. CNTK is Microsoft's open-source deep-learning framework, which stands for "Computational Network Toolkit." The library includes feed-forward DNNs (DotNetNuke), convolutional nets and recurrent networks, and offers a Python API over C++

code. [44] TensorFlow is an excellent choice for deep learning network development. TensorFlow is a free and open-source software symbolic math library and is also used for machine learning applications such as neural networks. TensorFlow was developed by the Google Brain team for internal Google use, and released on November 9, 2015. [66]

In the TensorFlow platform, there is a library called Keras, which joined Google's TensorFlow's core library in 2017. Keras is a high-level neural networks API, written in Python. It is designed to provide users with a user-friendly and higher-level set of interfaces that make it easy to develop deep learning models regardless of the computational backend used. It has support for many neural-network blocks such as layers, optimizer, and activation functions in various types of neural-networks, like standard neural-networks, CNN, and RNN. [64] Thus, Keras does a fine job in this experiment. Once the local computer system has a final model and a set of weights with adequate accuracy, they shall be embedded in the UWP application for sending UAV avoidance commands.

3.3 Source of Images

The images in this project are taken from different drones, and most of the views in pictures are taken outdoor in bright sunny days. Around 26,954 1920×1080 pictures were taken by a DJI Phantom drone or a DJI Mavic Air drone. The images from DJI drone were screenshots of the video it took and were labeled manually with different actions it should take for the purpose of supervised deep learning training. The obstacles in images are diverse as they are cars, trees, persons, electricity poles, buildings, etc., which are very commonly found in suburban environments.

All the images are divided into two sets, the training set, and the test set, with a ratio of 2.5:7.5. Each set is labeled into five categories: 'keep going', 'turn left', 'turn right', 'go up' ,and 'back up', for deep learning model development. The images in 'keep going' indicate that when the UAV receives an image like this, it can keep going straight. The images in 'turn left' and 'turn right' have the obstacle on their right or left, moving left or right helps it avoid the obstacle; the images in 'go up' give UAV a suggestion to climb up for a clearer vision and the reason why an image is assigned to 'back up' is not only for safety concerns but also the

(a) Turn Left

(b) Turn Right

(c) Back Up

(d) Go up

Figure 3.2: Same Object in Different Categories

possibility UAV might be trapped because the distance to the obstacle is too short to start an obstacle avoidance maneuver. With the visual situation divided into the five categories, the UAV can avoid obstacles on high or low altitude, because when it's hard to tell what the action should be taken, the UAV always backs up, until it knows how. The figures 3.2 show an object assigned to different categories.

To classify the images acquired from the UAV video shooting, two persons are involved in the image classification. How the two persons classify the images is that each of them is assigned with a part of the images to classify as the criterion mentioned above. After the classification is finished, the other person proofreads the classified images. Finally, they classify the images as they discuss. Theoretically, with more people involves in the classification program, less bias is there. However, due to the limitation of manpower, only two are involved. Thus, there inevitable bias in the classification.

3.4 Application

DJI Windows SDK is selected to create a customized UWP app for UAV supervising. DJI Windows SDK allows users to create a customized app to unlock the full potential of a DJI aerial platform. It has various features relates to flight control, camera control, and missions, for instance: pre-defined mission, high and low-level flight control, aircraft state through telemetry and sensor data, live video feed of what the camera sees, etc. These features are sufficient for this project. [58]

In this application there are two pages to be designed. One of them, called 'SetWaypoints', is used to create a waypoint mission, start a mission and show real-time location information on a map and in a text box. On the top of this page, there should be a panel displaying the UAV's real-time location, including latitude, longitude, and altitude. Below that panel, there are three text boxes letting users input the waypoint's locations, in the three dimensions, included in an expected mission. A real-world map with the current location of UAV on it would be displayed on the right half part of this page, and after a waypoint mission is created, the locations of every waypoint would be shown on the map as pins. Besides these components, there should be also several buttons, 'ground station mode', 'creating mission', 'stop the mission' and 'auto-landing', which are auxiliary functions for the UAV flight. The page is shown in figure 3.3.

The other page, called 'FPV', is designed for observing UAV's status. At the top of this page, a panel should show several UAV's current status info, including the velocities in the three dimensions. Also, the result and processing time of a loaded model would be display here. The rest of this page is assigned to a UAV first-person view (FPV) window, user can see the UAV's front view on this page. With the two designed pages, the users would have a great understanding of the situation of the aircraft and the prediction of a model; thus, the difference between the output of the model and the action it should take would be observed. Also, since the map shows the predefined mission waypoints and the UAV real-time location, the deviation of the course can be observed too. This page is shown in figure 3.4.

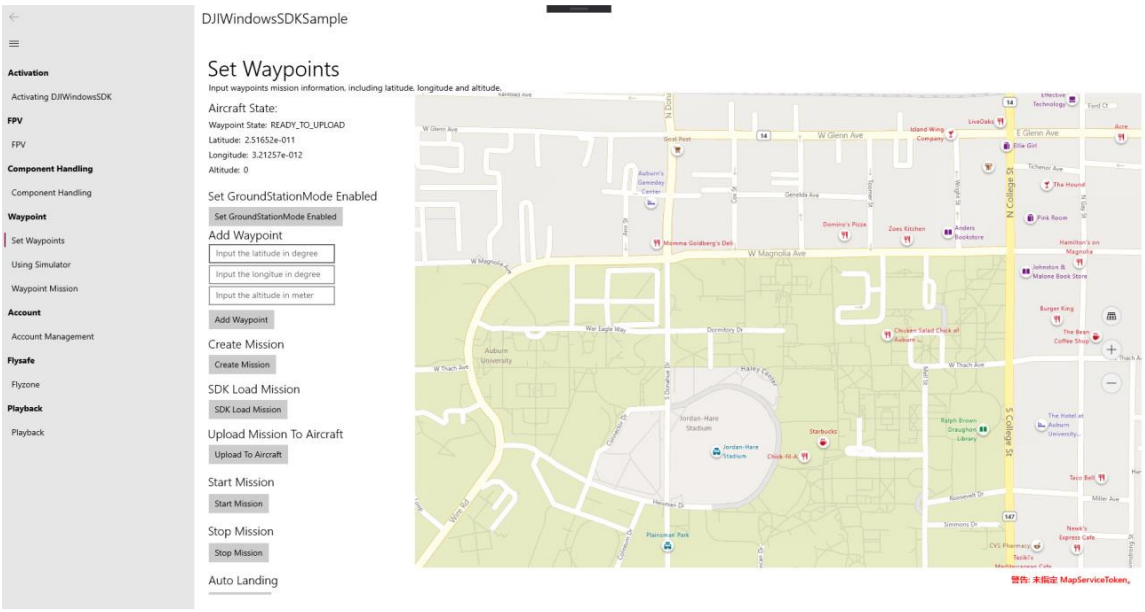

Figure 3.3: Set Waypoints Page

During a flight, the UAV onboard system doesn't use any time to learn anything. It only sends the images through a Wi-Fi connection to the ground station computer for the next moving direction request. After the application receives an image, it is input into the decision-making model; thus, the ground station knows what the next action is to send back to UAV. Besides sending the next action to UAV, the application also does path planning. Since there is a GPS receiver onboard, and the ground station know the waypoints (GPS coordinates) and the final point, it can do an excellent job on path planning, whatever the method is implemented.

3.5 Model Conversion

Since the deep learning models are created and trained in the Keras framework that is installed on a Linux system desktop; however, the Keras model is incompatible with UWP [17]. To implement the Keras model on the DJI UWP application, two different potential methods are tested.

First, because the model is developed in the Python language, embedding a Python file into a C++ environment is tested. Since a library file 'python.h' comes with Python language package, a python file can be easily run when it is called from a C++ program. The preparation work needed is importing 'python.h', linking Python package with the program, and setting up

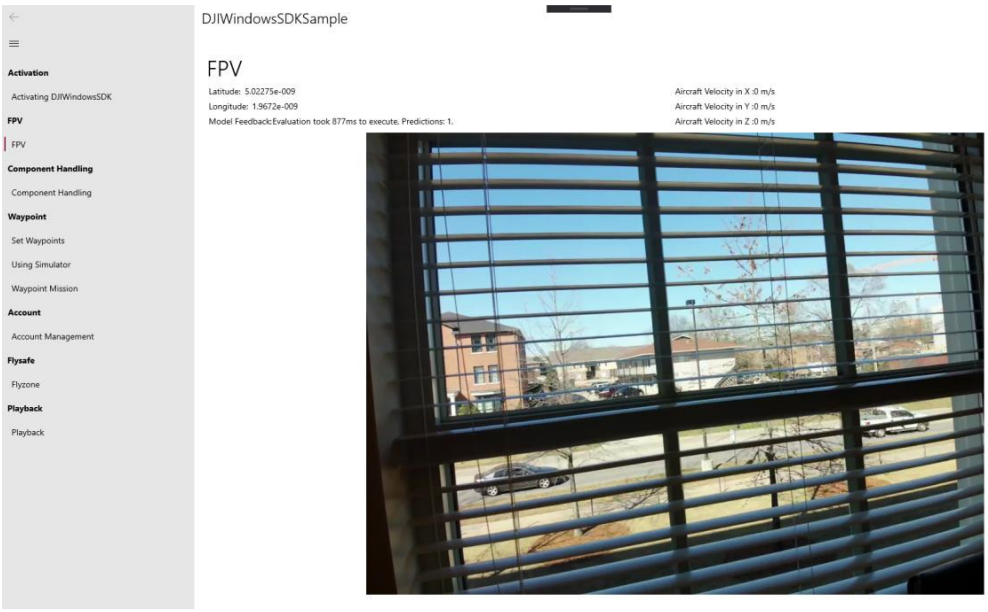

Figure 3.4: FPV Page

the variables as 'PyObject' type. For example, if the Python file is a function of calculating the sum of two input numbers, using a function 'Py_BuildValue' to build the input 'PyObject' variables. This method is pretty handy when Python file is work on a lightweight job. However, when the Python file is assigned with the job of loading a deep learning model, like a several hundred megabits file, UWP would crush; because the application is running in run-time, but the loading model action takes too much time to block other threads. So, this method is not suitable for the project.

Due to the high time cost of loading the model, loading a Keras model in a UWP application design page became more tempting. Windows AI is a tool published by Microsoft to allow developers to use trained machine learning models in Windows apps that are written in C# [18]. Importing the 'Windows.AI.MachineLearning' package grants the UWP application the ability to load the model directly on page design code and predict with a deep learning model using image resources taken from a UAV camera. What prevents Windows AI from loading Keras deep learning model is the fact that Windows AI is incompatible with a Keras model. However, thanks to the strong compatibility of the Open Neural Network Exchange (ONNX) model, Keras can be converted to the ONNX model to be loaded in the Windows AI environment, as figure 3.5 shows. There is an open-source library 'Keras2Onnx' on GitHub, which is a perfect

tool to convert a Keras model to an ONNX model [13]. The difference between the original and converted models with the same input is less than 0.01%. Thus, the model loading and prediction of images in the UWP application design page are both fast and suitable methods for this project.

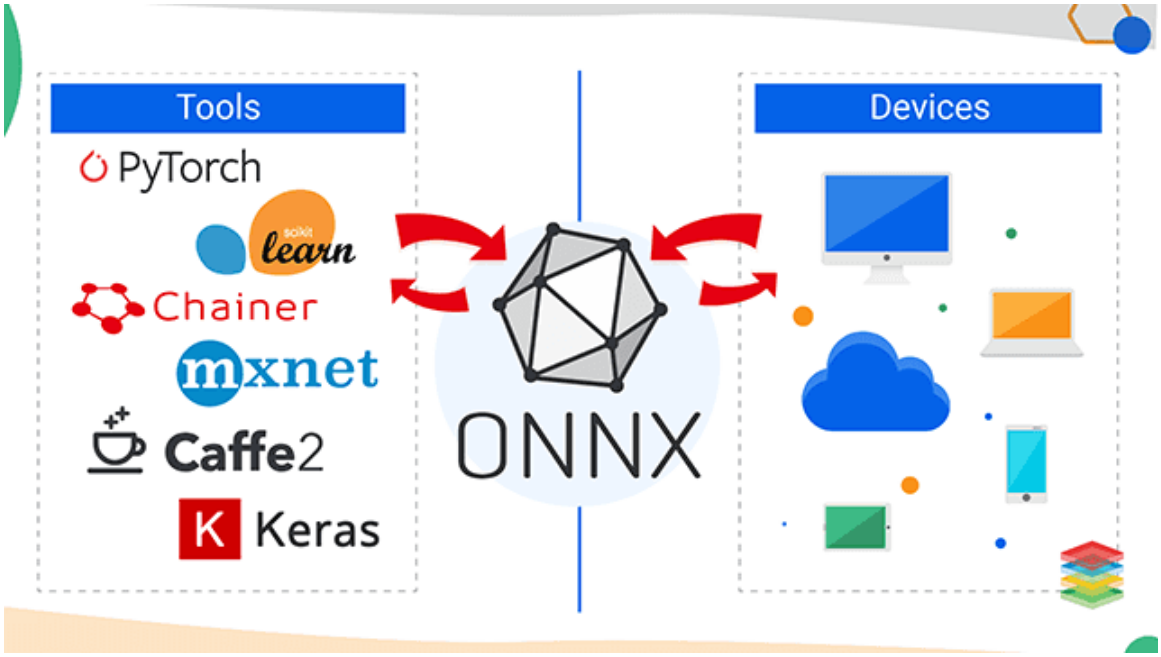

Figure 3.5: ONNX Compatibility [23]

Once the model from Keras is converted to an ONNX model, it is ready to be fed with a preprocessed image. Because an input images of a Keras model are normalized images, which indicates that the values in each channels are divided by 255 respectively. To preprocess an image before feeding it to a ONNX model, its values in each channel have to be normalized as well. Additionally, the input images of the Keras model default as RedGreenBlue(RGB) images; however, the default image format in Windows AI is BlueGreenRed(BGR). Thus, one more thing to do before inputting images into the ONNX model is reformatting the color channels.

Chapter 4

Deep Learning Models

Because of the wide application of deep learning, computer vision has been rapidly improving. There are several application areas that deep learning computer vision is now helping. Self-driving cars can figure out where the other cars and pedestrians are and to avoid them by using deep learning computer vision. Also, face recognition works better than ever before, so that people can unlock a phone or a door just using their face. Also, some apps are using deep learning to show the most attractive, the most beautiful, or the most relevant pictures users want to find.

4.1 CNN

There are several reasons why CNN is suitable for computer vision. First, the computation cost is much lower than a standard neural network. Here is an example: say we have a 1000 by 1000 RGB image and a standard neural network, it has 3 million input features which means the dimension of the input matrix is 3 million. Maybe there are 1000 hidden units in the first hidden layer. Then the weights matrix will be a 1000 by 3-million-dimension matrix. The computational and the memory requirements to train a neural network with 3 billion parameters is infeasible. However, with CNN, the number of weights that are trained is much smaller, around 200 in the example, because it only trains kernels of a 2D convolution operation whose sizes usually range from 3 by 3 to 9 by 9.

Additionally, we should consider parameter sharing. Parameter sharing is motivated by the observation that a feature detector such as a corner detector useful in one part of the image is probably also useful in another part of the image. That means a three by three filter for

detecting corners, with the same parameters, can be applied everywhere on the image. This is true for low-level features like corners, as well as the higher-level features, like detecting the eyes. So, maybe there is no need to learn separate feature detectors for the different part of the image; they're sharing feature detectors across the image. The feature may look a little bit different in different locations, but they might be similar enough.

Finally, there is the the sparsity of connections. The sparsity of connections means in each layer, each output value depends only on a small number of inputs. Here is an example, for a 3 by 3 corner detection filter; an output value is only relevant with 9 input pixels, not any of the others. With these two mechanisms, a convolutional neural network can have a lot fewer parameters than a regular neural network does.

Since this project needs a model to be fed with images and classify them, ResNet sound like a perfect fit. Very deep neural networks are difficult to train because of vanishing and exploding gradient problems. However, Residual Networks applying skip connections allow users to take the activation from one layer and suddenly feed it to another layer even deeper in the neural network. And by using this, building a very deep network is possible (sometimes it has even 200 layers [26]).

As a really deep neural network has potential flaws, why is a deep neural network still required in many ways? One of the reasons is that a convolution layer in a neural network works differently from a normal dense layer. If a computational result of a hidden unit is visually presented as an image, the image indicates what this hidden unit is looking for in an input image. In the first layer, maybe the images are some lines or dots with colors; however, in deeper layers, features or patterns even more complex is detected. Thus, the deeper a neural network is, the more complex the pattern it can detect. The model for this experiment must have the ability to detect complex features not only because the image it sees is complicated but also because generating a moving command from a single image without overfitting requires complicated computation [39].

ResNets are built out of something called a residual block, as shown in figure 4.1, which consists of a few sequential layers and a shortcut layer. So rather than following the main path, the information from the input layer can now follow a shortcut to go much deeper into the

neural network. What that means is that the output of this block is not just the non-linearity of information from the previous sequential layer, but also the information from the shortcut.

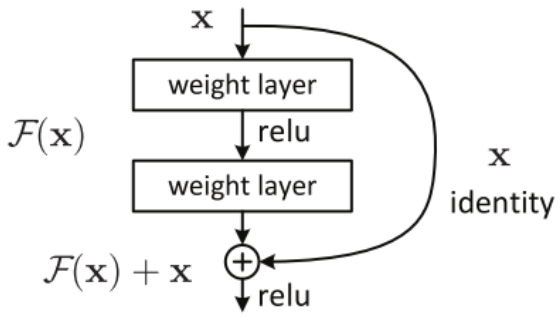

Figure 4.1: Residual Block [26]

Currently, there are two versions of Resnet, ResNetV1, and ResNetV2. The figure 4.2 shows the basic architecture of the post-activation (V1) and the pre-activation (V2) of versions of ResNet. ResNetV1 adds the second non-linearity after the addition operation is performed in between the input and the block output. ResNetV2 has removed the last non-linearity, therefore, clearing the path of the input to output in the form of identity connection. ResNetV2 applies Batch Normalization and ReLU activation to the input before the multiplication with the weight matrix (convolution operation). ResNetV1 performs the convolution followed by Batch Normalization and ReLU activation. The ResNetV2 mainly focuses on making the second non-linearity as an identity mapping, i.e. the output of the addition operation between the identity mapping and the residual mapping should be passed as it is to the next block for further processing. However, the output of the addition operation in ResNetV1 passes from ReLU activation and then transferred to the next block as the input. [23]

4.2 Time Series Model

4.2.1 RNN Models

Long Short Term Memory (LSTM) is an artificial RNN architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). [65] As shown in figure 4.3, X_t means the input of the current state, X_{t-1}

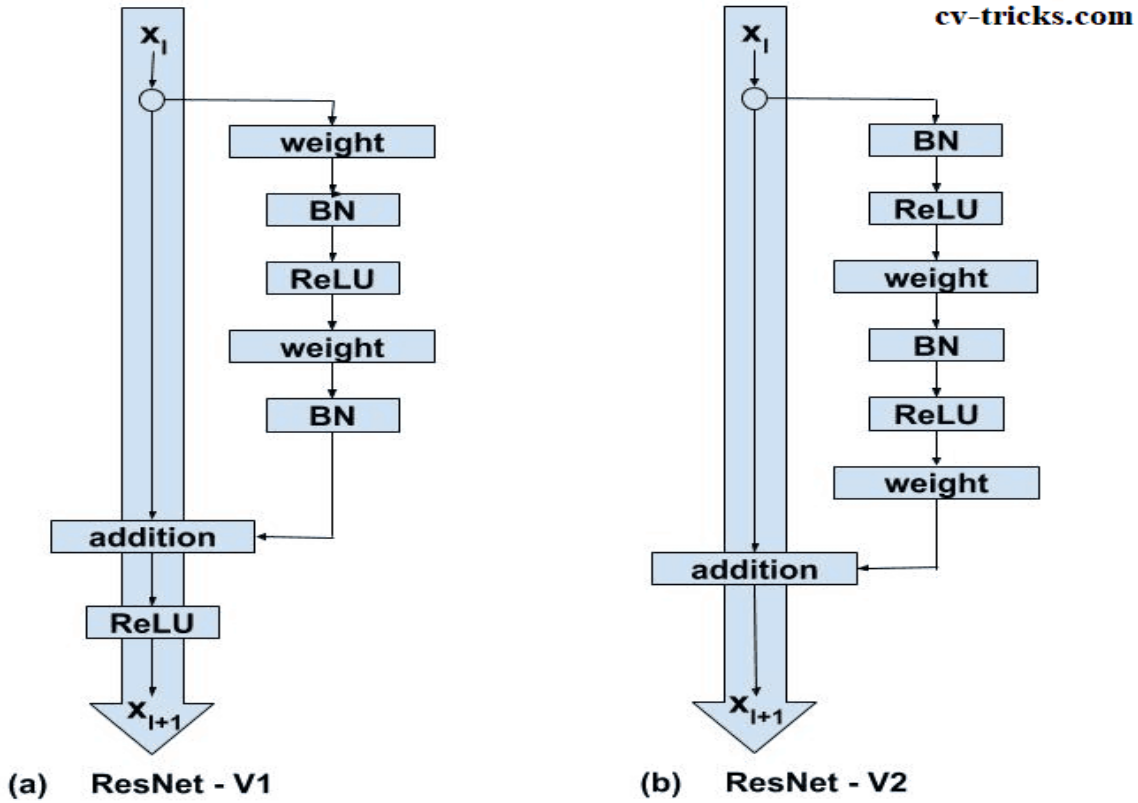

Figure 4.2: ResNetV1 vs ResNetV2 [23]

means the input of the previous state. The inputs of the LSTM unit are current input(X_t), cell state of the previous state(C_{t-1}), and output of the previous state(O_t, h_t). The first step of LSTM is to decide what info to keep in the cell state, and it is decided by a gate called 'forget gate', which is form by a sigmoid unit with two inputs: X_t and h_{t-1}. Output is a tensor that has the same size as $Ct-1$, and the values are either 0 or 1, which means forget or keep. At this step, some old info is thrown, like the type of the main object in the last scene which is not in the current scene. The second step is to decide what new info stores in the cell state. This is composed of two parts: one is called 'input gate', which is a sigmoid layer similar to the 'forget gate', outputs an either 0 or 1 vector, and the other one is a tanh layer, which generates a candidate vector to be stored. The output of these two layers is fused and then is updated with $Ct-1$. Finally, since it has the current cell state and the current input, the current state output can be made by an 'output gate'. A filtered version of cell state is used: the filter mask is produced by a sigmoid layer with two inputs, X_t and h_{t-1}, which decide what value in the cell

state it uses. Then the output of a tanh layer is filtered, which is the final output of the current state.

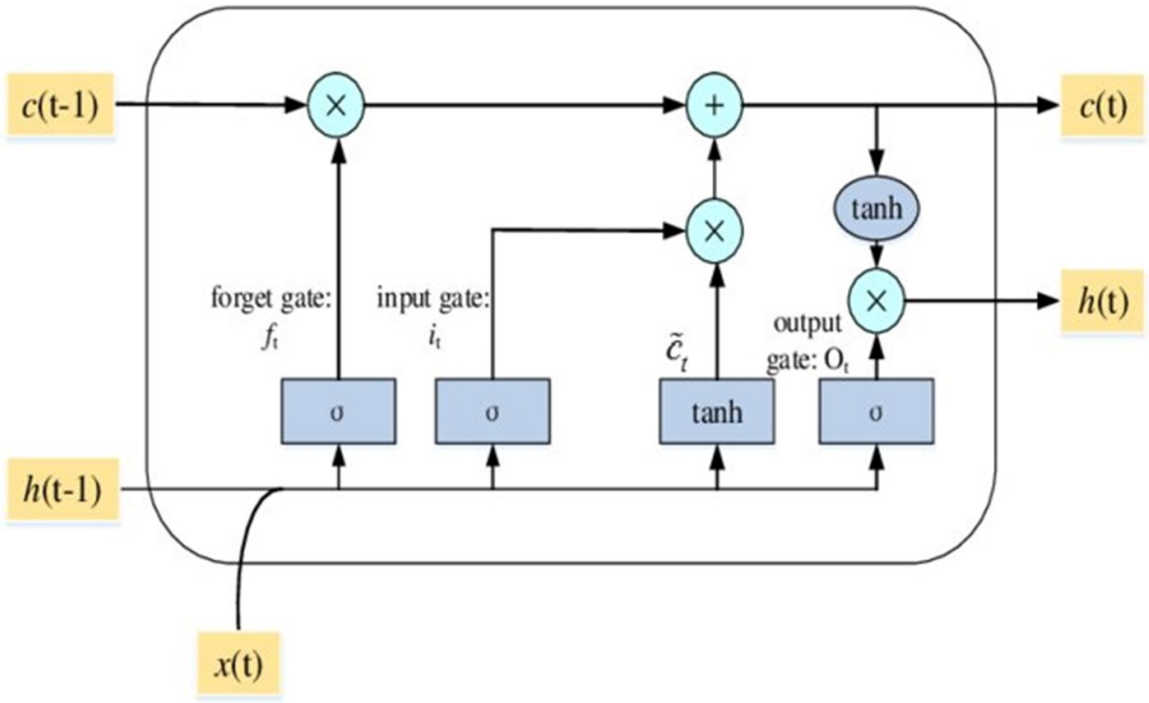

Figure 4.3: LSTM Recurrent Cell [70]

Different than LSTM, the Gat Recurrent Unit (GRU) [10] gets rid of the cell state but uses a hidden state to store the old info. As shown in figure 4.4, tt uses only two gates, 'reset gate' and 'update gate'. The 'reset gate' has the same function as the 'forget gate' in LSTM. It decides what old info to forget. It contains the same components as the 'forget gate' in LSTM but is applied on a hidden state instead of a cell state. The 'update gate' has both functions of the 'output gate' and the 'input gate'. It decides what new information to throw away and what new information to add to the hidden state.

4.2.2 A Proposed Simple RNN Model for Obstacle Avoidance

One possible reason for the poor accuracy of the ResNet family is that info from a single image can be limited, thus acquiring more info from UAV images seems to be helpful. Instead of absorbing features from a single image, using RNN to extract time-series can provide correlative info between consecutive images. Theoretically, the infomation of a new aspect in the input would improve the accuracy significantly, because now the model doesn't give its command

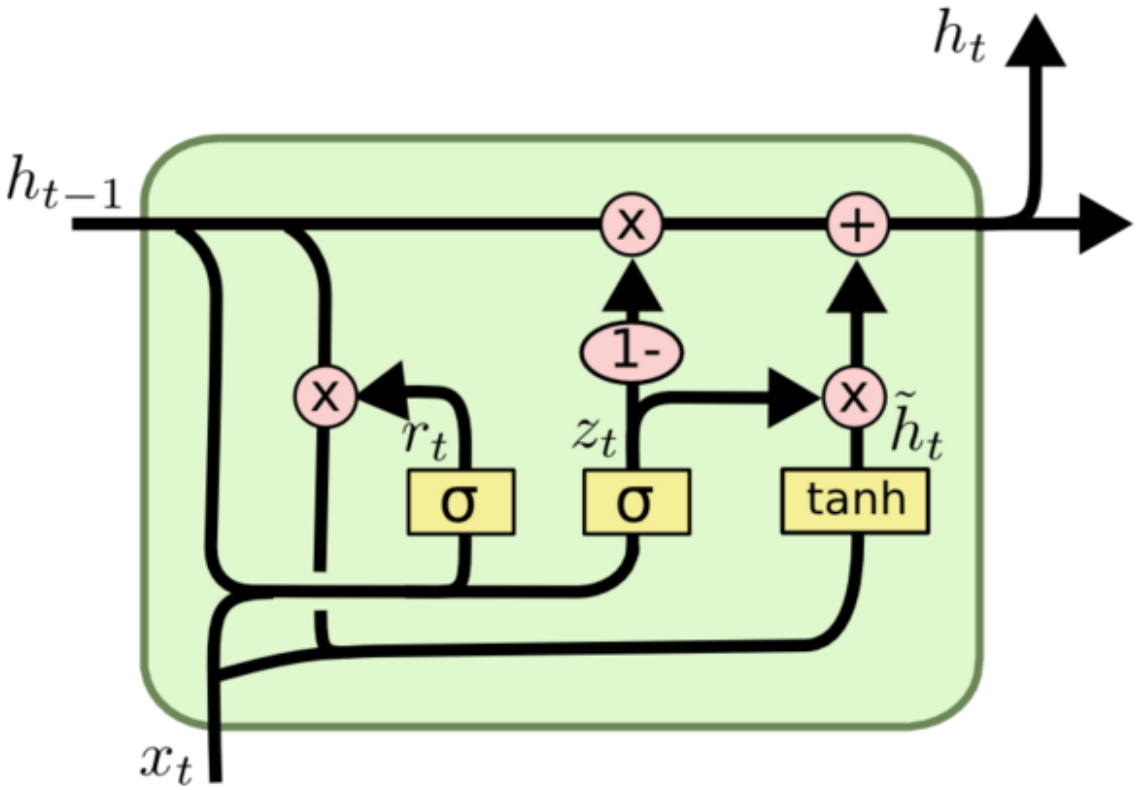

Figure 4.4: GRU Recurrent Cell [31]

according to only the objects' location in the image, but also the relative location to the last state or any previous state. With such implicit info, the model can guide the UAV command base on the changing of the object. For example, as a tree becomes larger and larger, which means the tree is approaching the UAV, then the command should be 'Back' or 'Up'.

To remedy the shortage of infomation on processing a single scenario in an image, a simple RNN model is designed as figure 4.5. To use this model, a set of consecutive images has to be made. For example, 5 consecutive screenshots snapped from a snippet of video. When the 5 consecutive screenshots are ready, they can be input into a ResNet, which is the model to be remedied. Then the output of the deep learning network is sent to the RNN module, it can be LSTM or GRU, successively, and the RNN module consequently have 5 corresponding outputs. Theoretically, using only the output of the last state should be enough, because RNN preserves the info from the original and intermediate state and refines them. However, to save more information from each state, the output of every output is utilized by inputting them into

a three-layers fully connected layer with a softmax layer to give the final obstacle avoidance command.

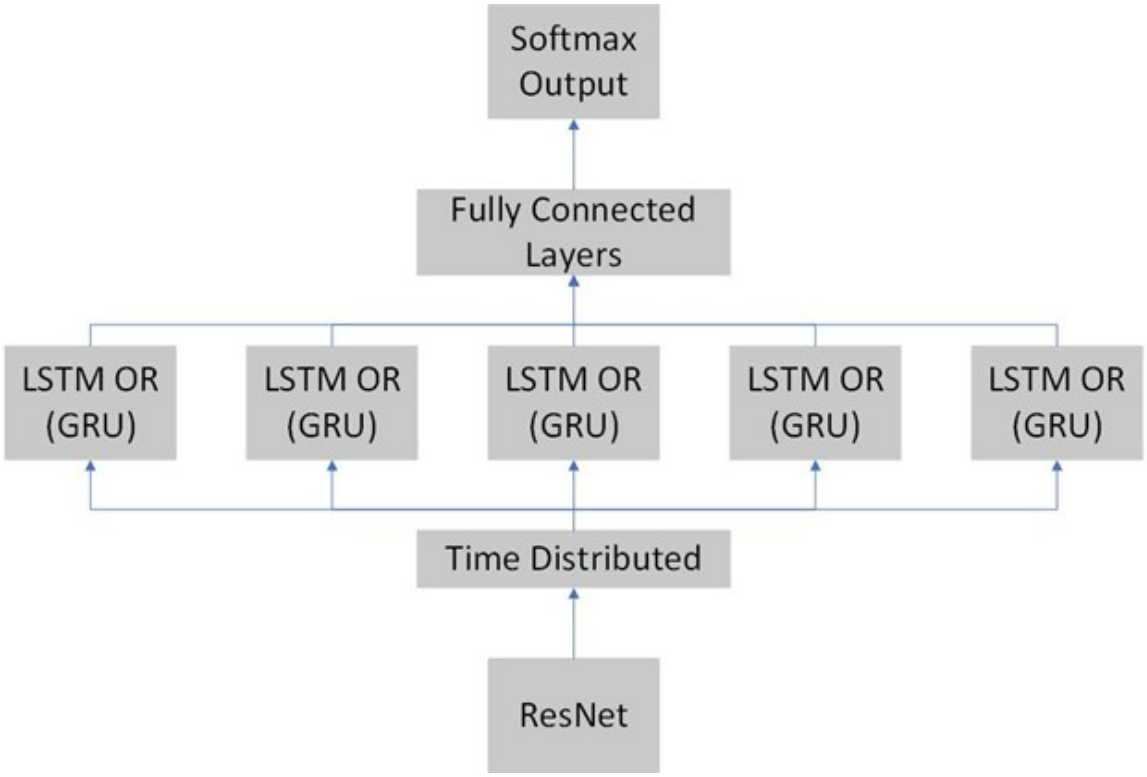

Figure 4.5: Proposed RNN Model

4.3 Data Augmentation

4.3.1 Traditional Data Augmentation method

Most computer vision machine learning programs require a lot of image data because computer vision is a difficult job. Normally, computer vision tasks must figure out what the pixels stand for and the positional relation between pixels. Complicated functions or say model structures are needed to accomplish the tasks; therefore, in practice, more data will help the model learn better with the complicated structures. So, to improve the performance of the training system, data augmentation, which can increase the number of the data from the original dataset, is a commonly selected technique. [62]

One of the data augmentation methods is horizontal mirroring, where the photo is flipped horizontally to get the object's left side to the right and the right side to the left. And for

most object identification or recognition tasks, if the object is a tree then mirroring it is still a tree. The only change is the position on the image. Another commonly used augmentation is random cropping. Random cropping means what it says: randomly cropping the image to a smaller size, which is then upsampled. There is a chance that the target objects are damaged in each cropped image. Therefore, random cropping is not a perfect data augmentation. There is another type of data augmentation called color shifting. This technique does not add any distortion to pixels' positions, instead, it adds distortions to color channels, for instance, adds 10, 20, and -30 to Red, Green, and Blue channels, respectively. One of the benefits of this method is that the users can adjust the sunlight in the images. It can be brighter, yellower, redder, etc. However, no matter how the color is distorted, the label or say the identification of the images remains the same. This data augmentation method can make the model more robust to changes in the color of image data.

4.3.2 GANs Data Augmentation method

GANs can play a part in data augmentation. Since sometimes the real data can be too expensive to get and there is insufficient data, some supplemented synthetic data can be adopted to increase the size of the dataset. GANs are suited for this task because they can generate fake images to fool the models like a classifier, detector, or segmentation model. Unlike the flipping, rotating, or zooming in traditional data augmentation techniques, GANs provide a variety of new data. For example, if the task of a classifier is to classify different kinds of animals, and one of the data is an image of a black short-hair cat, with traditional data augmentation techniques, the cat in the images remains the same. However, with the GANs, you will get different kinds of cat, such as a white long-hair cat. Furthermore, because the generated image can have various backgrounds, object orientations, object positions, etc., it shows the features of traditional data augmentation techniques.

There are pros and cons of GANs data augmentation. [1] One of the pros is that the GANs generated image is usually better than the handcraft images. It is more realistic. Another pro is the GANs can generate more labeled examples. That is awesome because if the training dataset is imbalanced or it does not have enough examples of a certain class, conditional GANs can

generate significantly more labeled examples for those classes. The third pro is that the GANs can generalize the downstream model, like a classifier. For example, the number of images in the class 'benign tumor' is much lower than those in 'malignant tumor', and there is no way to get more example images from the hospitals. The GANs can generate more data that mimic the 'benign tumor' images to make the dataset more balance and the model to understand classification better.

The GAN data augmentation is not perfect, there are several cons of it. First, it does not cover the entire diversity of the target class if the training data is limited. In other words, the diversity of generated images still relies on the diversity of training dataset. Related to that issue, when the GANs models starts to memorize the training data, the generated images look almost as same as the original image. This would help the downstream model because if the fake images are looked identical to the real images, it is just like training for another epoch instead of learning from them.

4.3.3 Data Augmentation with GauGANs

As for selecting a suitable data augmentation method for this project, GANs are the winner for various reasons. Traditional methods include flipping, rotation, cropping, etc., but, any of these techniques may change the position and size of the objects. In classification tasks, the changes do not affect the system but bring diversities to the dataset. In this project, traditional data augmentation techniques lead to changes of the labels of images. Since the GANs can generate vivid synthetic image without changing the object's position, a tree in the real images would be transformed into a tree in a different appearance with the same position and size in the fake images, it keeps the feature information as the original images. Therefore, the GANs increase the diversity of the dataset without scarifying the information. [15] [45]

In GANs category, there are many different extensions, for instance, Conditional Generative Adversarial Network (cGANs) [41], Stacked Generative Adversarial Network (StackGAN) [28], Pix2Pix [30], etc. The input to both generator and the discriminator models of the cGAN is composed of the information in addition to the image. For example, labels or latent space can be used as input. StackGAN generates images from text using a hierarchical

stack of conditional GAN models. The architecture is constituted of a series of text and image-conditional GAN models. The first level generator is conditioned on the text and generates a low-resolution image. The second level generator is conditioned both on the text and on the low-resolution image output by the first level and outputs a high-resolution image. Since the generated images should preserve the position, size, and kind of objects, the GANs described above do not fit the task.

The Pix2Pix model is an image-to-image generation model. In this model, the U-Net model architecture is used in the generator model, and the PatchGAN model architecture [30] is used in the discriminator model. The loss of the generator model will be updated including the vector distance from the fake image to the target output image. There are several extension models to Pix2Pix GAN, like Cycle GAN [74], Style GAN [34], GauGANs [47], etc. Unlike Cycle GAN or Style GAN, which changes the whole subject of the image, GauGAN can keep the features well, this GAN model is chosen to generate synthetic images.

The GauGAN that does semantic image synthesis with SPADE (Spatially-Adaptive Normalization) [48] have an impressive result of turning semantic images into synthetic images, and it outperforms other popular synthetic image generating models, such as cascaded refinement network (CRN) [9], semiparametric image synthesis method (SIMS) [51], and pix2pixHD [61]. Semantic images are mask images that contain the location coordinates and class labels of every object, and with the info been kept in semantic images, they can accurately reproduce the original image with the same object class labels. GauGAN accepts various renowned semantic segmentation datasets, such as the COCO-Stuff dataset [2], ADE20k [73], and Cityscapes [11]. Since the COCO-Stuff dataset has the highest percentage of users who favor the results of this method over those of the competing methods, it is chosen to be the semantic segmentation and annotation benchmark. Furthermore, another advantage of the COCO-Stuff dataset is it has 164k images which outnumber others, and the images are not a constraint to specific categories, for instance, city scenes like the images in the Cityscape dataset; therefore, a well-trained COCO-Stuff based model can provide a more generic result.

The only thing that is required to use GauGAN is a method to create semantic segmentation images. There is a constraint of the technique, that the labels of the recognized

objects must be the same as COCO-Stuff data set class labels. DeepLabv2 [6] is used for this purpose, because of its good performance compared to other concurrent semantic image segmentation models, such as DeepLab-CRF-LargeFOV-COCO [6] and Adelaide-VeryDeep-FCN-VOC [67], and the compatibility to COCO-Stuff data set, including object position info and label names. Thus, after converting images to semantic mask images and an annotation file, which contains detailed info of the objects that belong to COCO-Stuff classes, GauGAN can generate the desired synthetic images. It is safe to say that the generated images keep the object's kind and position. The figure 4.6 and the figure 4.7 show an original image and a synthetic image generated by GauGAN.

Figure 4.6: Source Image

4.4 Semantic Segmentation and Instance Segmentation

Early object detectors were based on a sliding-window-based approach which was computationally inefficient and poorly accurate; however, modern techniques include many deep-learning-based algorithms. There are several different types of Computer Vision algorithms, such as object detection, semantic segmentation, instance segmentation, etc. Each has unique architecture and different tasks to perform on the input images. The outputs of these models are potentially beneficial to the obstacle avoidance network, and the specific details are below.

Figure 4.7: GauGAN Synthetic Image

4.4.1 Computer Vision Tasks

Object Detection

The object detection models' task is identifying the objects and correctly labeling them. These tasks decompose into two steps: object localization and image classification. Object localization refers to a method to find the location of the object and produce the tightest possible a bounding box around the object. Image classification is simply feeding the localized objects to a classifier that labels them. Thus, an object detection model usually has a regression module and a classifier module, which produce the bounding boxes and the labels, respectively. [72] The produced info includes the type of objects and where they are located in the image. More information can be inferred with them, such as the approximated distances between the camera and the objects, and this could be beneficial to the obstacle avoidance model.

Semantic Segmentation

Semantic segmentation focuses on localizing and labeling objects in the images as well. However, it does not provide the bounding boxes and the labels of objects but generates a label map for an individual input image. On the label map, each pixel is linked to the corresponding pixel

on the input image by the label in the label list. For example, a pixel value 128 on coordinate X=50, Y=50 indicates the pixel X=50, Y=50 on input image is one of the pixels consist the object NO.128, which is 'House' in Common Objects in Context-Stuff (COCO-Stuff) dataset. Since the output is a label map with the same size as the input images, the model can be an auto-encoder or end with a convolutional layer. Similar to the object detection model, this algorithm provides the label and the location of the object; thus, this technique will benefit the obstacle avoidance model.

Instance Segmentation

Instance segmentation associates a class label to each pixel like semantic segmentation, except that it treats multiple objects of the same class as individual objects. [24] The output of an instance segmentation models consists of regions of interest (ROI), class labels, and label masks. Label masks are binary matrices that indicate the specific location of the object pixel-wisely, and the ROI tells the bounding boxing of the objects. These are like a combination of object detection and semantic segmentation; however, they can only label the object with the class number, not label every instance. So, the output of instance segmentation should benefit the obstacle avoidance model more than those techniques.

4.4.2 YOLO

A good way to get accurate bounding boxes of objects is with the YOLO algorithm. What this algorithm does is first it places down a grid on the input image, then detects objects on each grid. Every grid has an output of vectors. There is a number that indicates if there is an object; if so, four numbers show the location of the object and the number of preset class tells what the object is. Of course, more grids mean higher object detection accuracy; however, the grid's size affects computation cost too much: the larger it is the more cost there is [52]. The model is shown in figure 4.8.

What if the algorithm detects multiple objects in one grid? Using the idea of anchor boxes can help it a lot. With anchor boxes applied on all grids, they now have not only one vector but the number of anchor boxes vectors or one vector with the size of that number by one vector

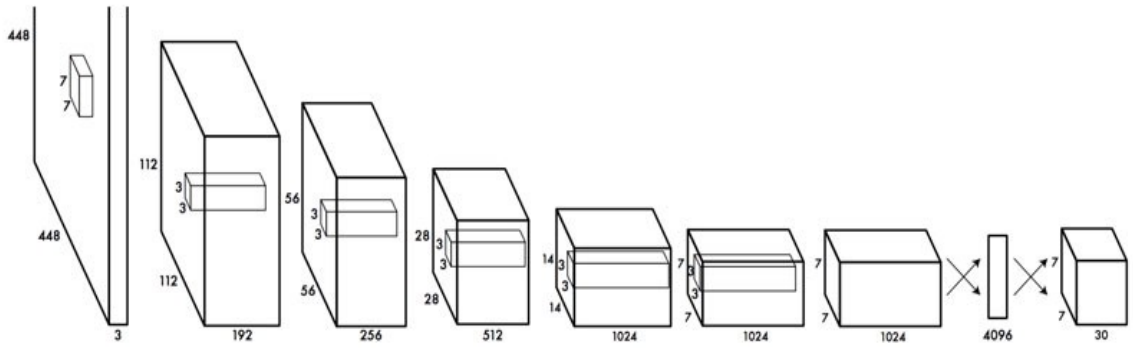

Figure 4.8: YOLO Model [52]

size. After having a multiple number of boxes, the algorithm uses Intersection over Union(IoU) and Non-max Suppression to select the box of class with the highest possibility.

For example, say a 19 by 19 grid is applied on the image, two anchor boxes are preset to be detected and the algorithm is trying to detect 3 objects, so the vector size of a grid is 16. Since there are 19 by 19 grids, the size of the final matrix is 361 by 16 or 19 by 19 by 2 by 8. Since two anchor boxes are set, each of the grid cells gets two predicted bounding boxes. Some of the bounding boxes can go outside the height and width of the grid cell that they came from. Which bounding box belongs to which anchor box? The bounding box has the highest IoU with an anchor box becomes a pair. Next, we get rid of low probability predictions by applying the Non-max Suppression algorithm. Finally, for each of the three classes, we independently run Non-max Suppression for the objects that were predicted to come from that class. So that's it for the demonstration of the YOLO object detection algorithm, which is one of the most effective object detection algorithms.

There are several version of the YOLO algorithm at present: the original version (v1), the second version (v2) and the third version (v3). YOLOv1 uses the Darknet framework which is trained on the 'ImageNet-1000' dataset. This works as mentioned previously but has many limitations, and because of it the use of the YOLOv1 is restricted. It could not find small objects if they appear as a cluster. This architecture found difficulty in the generalization of objects if the image is of other dimensions different from the trained image. The major issue is the localization of objects in the input image. [29] [33]

YOLOv2 has the following changes. It normalizes the input layer by altering slightly and scaling the activations. By adding batch normalization to convolutional layers in the architecture, MAP (mean average precision) has been improved by 2% [12]. It also helped the model regularise and overfitting has been reduced overall. The input size in YOLOv2 has been increased from 224*224 to 448*448. The increase in the input size of the image has improved the MAP (mean average precision) up to 4% [56]. One of the main issued that had to be addressed in the YOLOv1 was the detection of smaller objects in the image. This has been resolved in the YOLOv2 by dividing the image into 13 by 13 grid cells which are smaller when compared to the previous version. This enables YOLOv2 to identify or localize the smaller objects in the image and is also effective with larger objects. [56] YOLOv1 has a weakness detecting objects with different input sizes which says that if YOLO is trained with small images of a particular object it has issues detecting the same object with a bigger size. This has been resolved to a great extent in YOLOv2 where it is trained with random images with different dimensions ranging from 320 by 320 to 608 by 608 [56]. This allows the network to learn and predict the objects from various input dimensions with accuracy. YOLOv2 uses the Darknet 19 architecture with 19 convolutional layers and 5 max pooling layers and a softmax layer for classification objects [53]. [33]

The previous version has an incremental improvement which is now called YOLOv3. As many object detection algorithms have been around for a while now, the competition is all about how accurate and quickly objects can be detected. YOLOv3 has all we need for object detection in real-time with accuracy and classification of the objects. [54] YOLOv3 gives the score for the objects for each bounding box. It uses logistic regression to predict the objectiveness score. YOLOv3 uses logistic classifiers for every class instead of softmax which has been used in the previous YOLOv2. By doing so in YOLOv3 we can have multi-label classification. With a softmax layer, if the network is trained for both a person and man, it gives the probability between person and man let's say 0.4 and 0.47. The independent classifier gives the probability for each class of objects. For example, if the network is trained for a person and a man it would give the probability of 0.85 to person and 0.8 for the man and label the object in the picture as both man and person.YOLOv3 makes predictions similar to the Feature Pyramid Networks

(FPN) [38] where 3 predictions are made for every location the input image and features are extracted from each prediction. By doing so YOLOv3 does better at different scales. YOLOv3 uses the Darknet-53 network for its feature extractor which has 53 convolutional layers. It is much deeper than the YOLOv2 and also has shortcut connections [53].

4.4.3 DeepLab

DeepLab is a state-of-art deep learning model for semantic image segmentation, where the goal is to assign semantic labels (e.g., person, dog, cat, and so on) to every pixel in the input image as shown in figure 4.9. [6] In DeepLab, the idea of Atrous convolution is introduced. Compared to standard convolution, Atrous convolution densely convolves the feature with an argument that represents the wideness of the field instead of convolves all the adjacent feature pixels. As in figure 4.10, the top figure shows a standard convolution, and the bottom figure shows an Atrous convolution. Atrous spatial pyramid pooling (ASPP) in figure 4.11 is used to robustly segment objects at multiple scales with filters at multiple sampling rates and effective fields-of-views. ASPP is an Atrous version of spatial pyramid pooling (SPP), parallel Atrous convolution with different rates applied in the input feature map. As objects of the same class can have different scales in the image, ASPP helps to account for different object scales which can improve the accuracy. The output of the DeepLab model is a semantic mask, which has pixel values to be class labels.

4.4.4 R-CNN

R-CNN as shown in figure 4.12, which stands for 'Region-Based Convolutional Neural Network', was debuted in [20]. This model might be one of the earliest large and effective convolutional neural networks to localize, segment, and classify objects on images. There are three modules that are proposed, a region proposal, a feature extractor, and a classifier. The model extracts around 2000 region proposals from an input image, then sends the proposals to a large CNN to get the features for each proposal, and thereafter, the classifier outputs the class of each region. For the region proposal stage, a computer vision technique called 'selective search' is used to propose candidate regions of potential objects in the image. Since window-slide-based

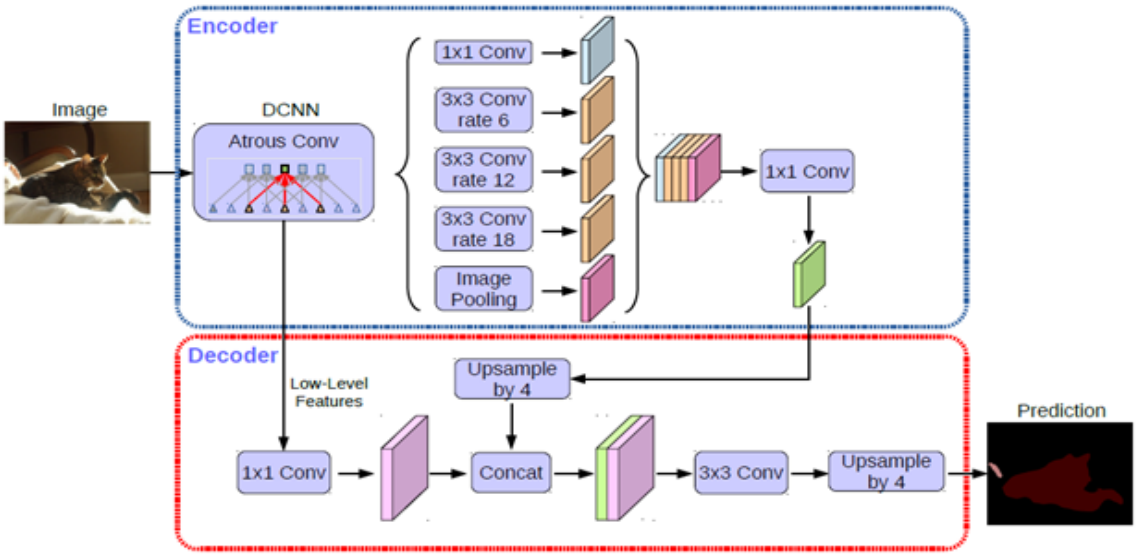

Figure 4.9: DeepLabv2 Model [7]

selective search groups regions that have similar color, texture, shape, and size, it can be very slow. At the feature extraction stage, the model outputs a 4096-dimensional feature vector by applying a CNN (usually an AlexNet) implementation on each regional proposal. Finally, the classifier can be a linear Support Vector Machine(SVM) that provides a label of the region proposal. The drawbacks of R-CNN are the following: training speed is very slow, each image has around 2000 region proposals before feature extraction, and each module in R-CNN has to be trained separately. [50]

4.4.5 Fast R-CNN

Fast R-CNN 4.13 was proposed in 2015 as an extension to speed up the training efficiency of R-CNN. [21] It adopts a single model instead of a multi-stage pipeline and provides the region and the class of the objects directly. The input of the model is the region proposals of the image connected to a layer called Region of Interest Pooling Layer (ROI Pooling) [21] which extracts the features of the corresponding region proposal. Then the features are fed to a fully connected layer and the output is separated into two outputs. One is the class prediction from a softmax layer, and the other one is the bounding boxes coordinates from a regressor layer. Each region proposal has a corresponding pair of outputs. The training speed is significantly

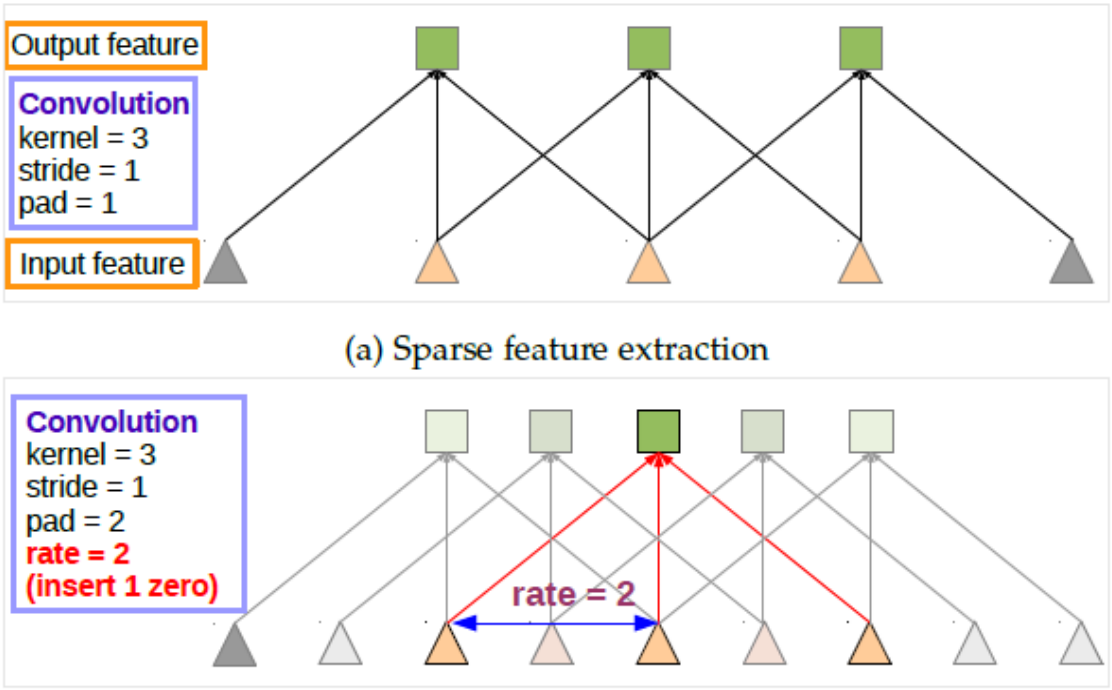

Figure 4.10: Atrous convolution [6]

improved compared to R-CNN, however, it requires a set of region proposals for each of the input images.

4.4.6 Faster R-CNN

A more advanced model that improves both the detection and training speed was proposed in [55]. Faster R-CNN improves the speed because of an additional module to the basic Fast R-CNN, called Regional Proposal Network (RPN) as shown in figure 4.14. RPN can refine and propose region proposals as a part of the training process. In RPN, the concept of anchor box is introduced, each feature map pixel has the number of anchor size multiplied by the number of anchor ratio boxed. For example, when the anchor sizes are 128, 256, and 512 and the anchor ratios are 1:1, 1:2, and 2:1, there are 9 anchor boxes for each feature map pixel as shown in figure 4.15. The region proposals provided from RPN have two attributes, the location of the box, and a score that indicates the confidence of the object is a foreground object. Assuming the feature map is 40 by 60 and each pixel has 9 anchor boxes, there will be around 20k region proposals for one single input image. To speed up the training and inference speed, lowering

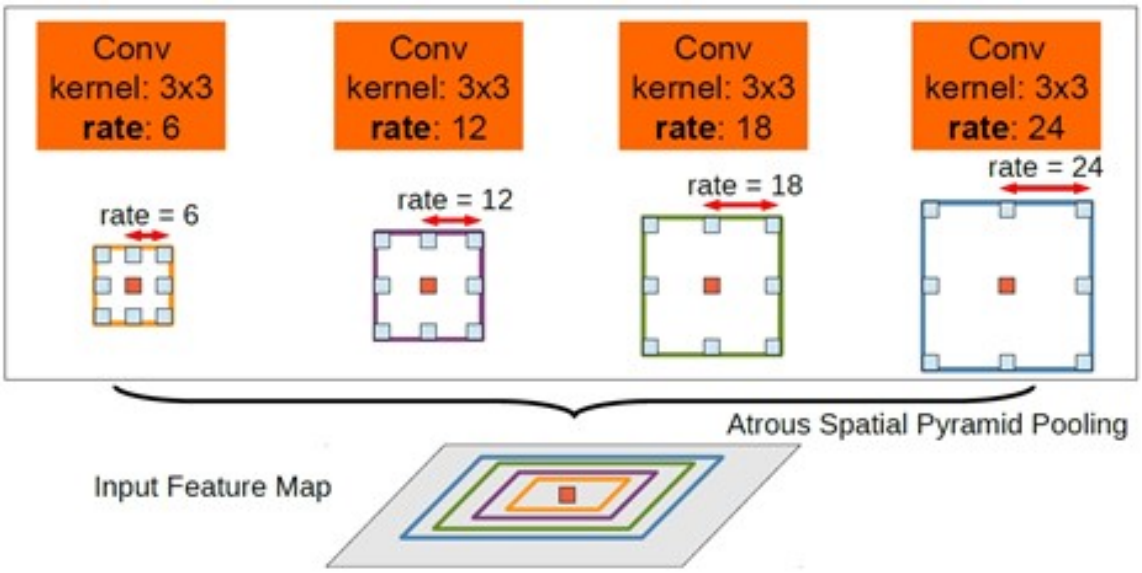

Figure 4.11: Atrous Sptial Pyramid Pooling [6]

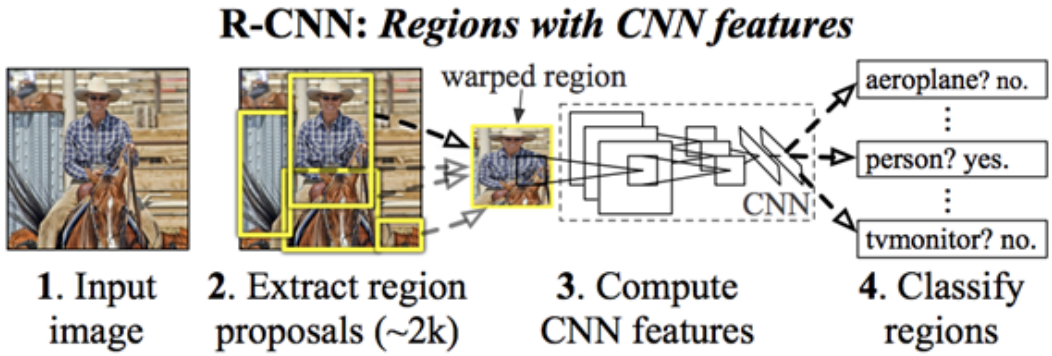

Figure 4.12: R-CNN Model [20]

the number of region proposals is necessary. The number of region proposals can be reduced by applying Intersection over Union (IoU) and Non-Maximum Selection (NMS) [55] as shown in figure 4.16. During the training process, the NMS algorithm eliminates those proposals that have low confidence score from the list by applying the following rules: firstly, choose the proposal with the highest score from the proposal pool; secondly, calculate the IoU score of the selected proposal with all other proposals; then discard those proposals that have lower IoU score than the threshold; finally, repeat the process until the pool is empty. IoU compares two proposed regions by simply dividing the union area with the intersection area. A higher score means the proposed region has more overlaps with the selected proposal area. With the

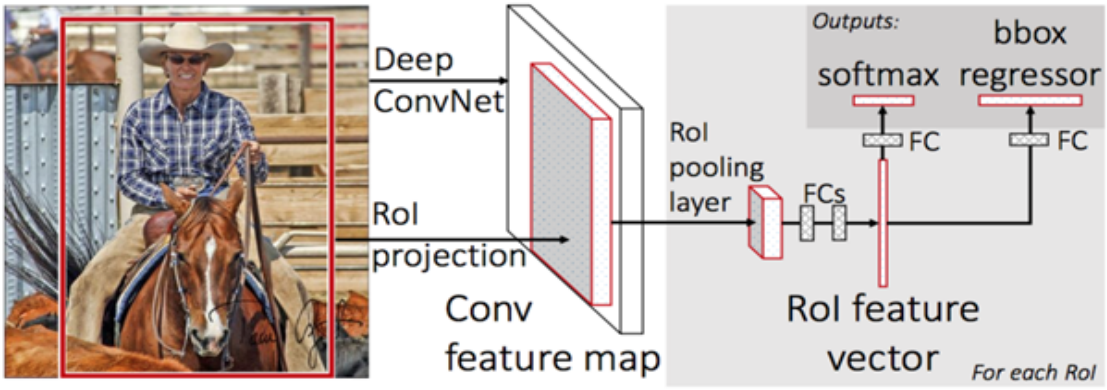

Figure 4.13: Fast R-CNN Model [21]

NMS and IoU layer, the number of RPN's outputs can be tremendously reduced. After each of the corresponding regions is extracted from the original image, they need to be fed to a fully connected layer; however, the uncertainty of the region shape prevents this application. To address this issue, ROI pooling is introduced. Firstly, the ROI pooling layer finds the areas on the feature map for each corresponding RPN output proposal, then the region is evenly divided into four parts. Finally, the highest pixel in each part forms a matrix that has a fixed size of 4. So, the output can be fed to the basis of Fast R-CNN and get the class label and bounding boxes. The figure 4.17 shows the Faster R-CNN model.

4.4.7 Mask R-CNN

Mask R-CNN [25] is the most widely used instance segmentation model so far. As a member of the R-CNN family, it is derivative from Faster R-CNN. It keeps the output of Faster R-CNN, which are bounding boxes and class labels of objects, and it also outputs a mask image that indicates each object instance in the input image. Based on the Faster R-CNN architecture, it simply connects a 4 layers convolutional neural network to the output of the ROI pooling layer in Faster R-CNN. 4.18 The output size of the model should be the same as the input image, and they are pixel-wise mapped. 3 pieces of info are output from Mask R-CNN, they are ROI bounding boxes, the class label of the object, and the instance mask. Therefore, this provides an improvement to the obstacle avoidance system because it contains more info.

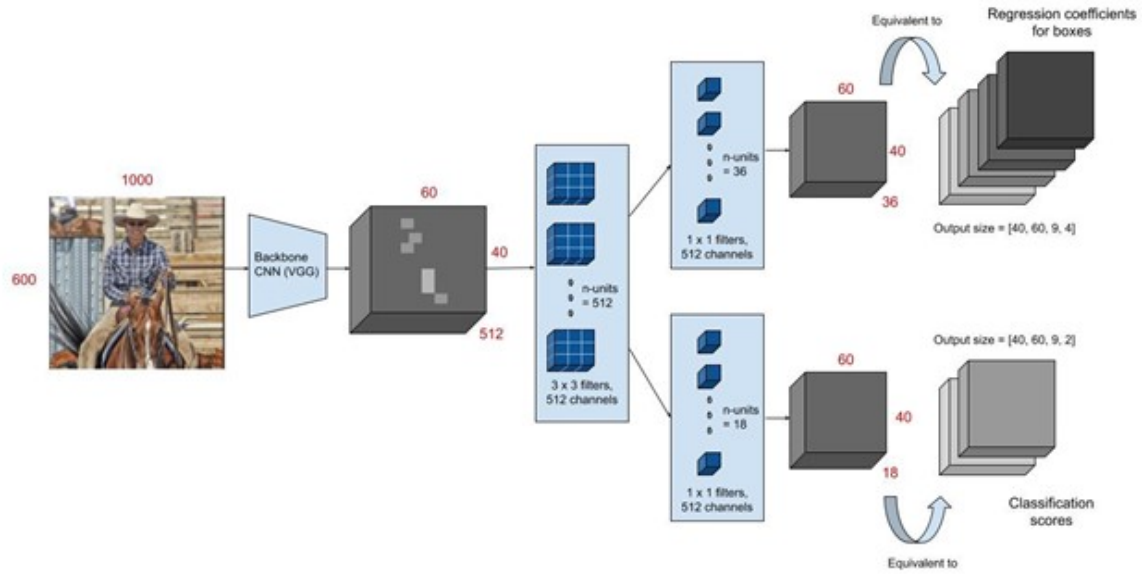

Figure 4.14: RPN

4.4.8 Two-Stages UAV Obstacle Avoidance Deep Learning model

Since the two models at the first stage provide different types of rich information of the image, semantic, and instance map respectively, the model composed at the second stage should have the ability to handle their outputs. Inspired by [32] , where the model has an acceptable accuracy with two inputs of the depth map and original image, a model that consists of some two-dimensional convolution layers and fully connected layers is shown in [32].

The designed deep learning model to learn the semantic map has the following input layers: image input and mask input. The image input layer catches 256 by 256 by 3 RGB-colored images and the mask input layer catches a semantic mask of the same size. The architecture of the proposed Deeplab-followed is shown in 4.19. Similar to [32], both inputs have many 2D convolutional layers that follow. For the first convolutional layer, the kernel size is adopted as 3 by 3, and the channel number is 256. Hopefully, the small kernel size and a large number of channels can extract many kinds of features. For the second convolutional layer, the kernel

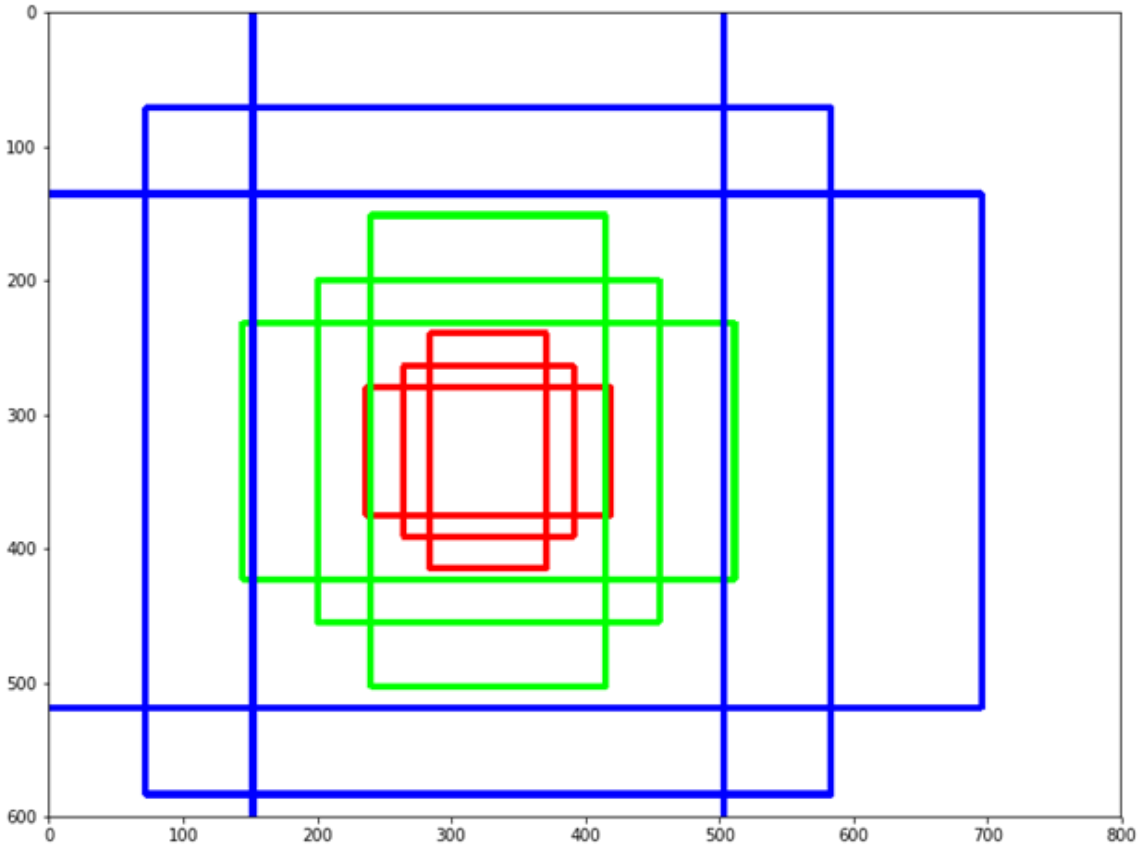

Figure 4.15: AnchorBoxes

size and the channel number are adopted as 5 by 5 and 128 respectively. For the third convolutional layer, its purpose is not to learn the features but to flatten the feature maps to a 1D tensor. Similar to the 2D convolutional layer, the layers that follow the semantic input have the same hyperparameters. For the second and the third convolutional operations, max-pooling of size 2, batch normalization, and ReLU activation is performed. Thus, the size of each feature map is quartered at the end of the third convolutional layer. From the different paths of the image input and the semantic mask input, the outputs are both a vector of length 64. To fuse the two features of the image and the semantic mask, a concatenate operation is applied. Thus, the size of the feature map is doubled and increased to 128. The following layers are two fully connected layers, which have a size of 1024 and 512 respectively, and a softmax output layer.

Different from the model that determines the semantic information, the model taht follows MaskRCNN has more input layers, since constructing the instance map requires more information than just a mask, but the ROI, class ID, and score of the prediction. This information

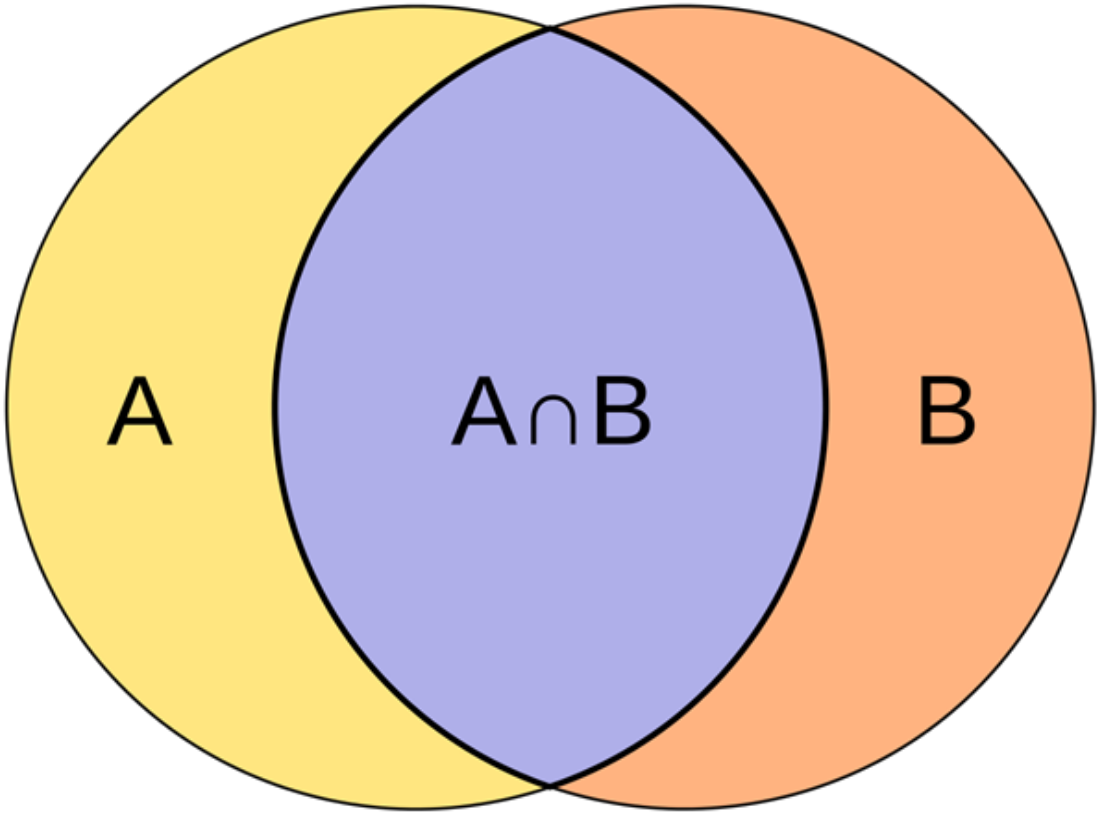

Figure 4.16: IoU

can be converted to various feasible data to be sent into an input layer: ROI is defined by four numbers which are the boundaries of the ROI square, thus ROI can be input as the shape 4 by 1; class ID is just a number from 0 to 91, each represents an object type of the target object; a score is also a number but from 0 to 1, represent the possibility of the target ROI has an object; mask is the instance mask which has a dimension that same as the original image and the pixels are either 0 or 1 that represents whether the corresponding coordinate has the target object, thus the shape of the mask input layer is 256 by 256. Similar to the paths in the model that follows DeepLab output, the model that follows MaskRCNN has the same structure on the paths of the image input and the mask input. The only difference is the layer channels are 128, 64, and 32 for the three 2D convolutional layers. To simplify the input data, the class ID and the score from MaskRCNN are fused into a vector of size 91. The index represents the type of object, this is what class ID does, and the number between 0 to 1 on the index represents the possibility of the object. On the paths of class-score input and ROI input, there are only two

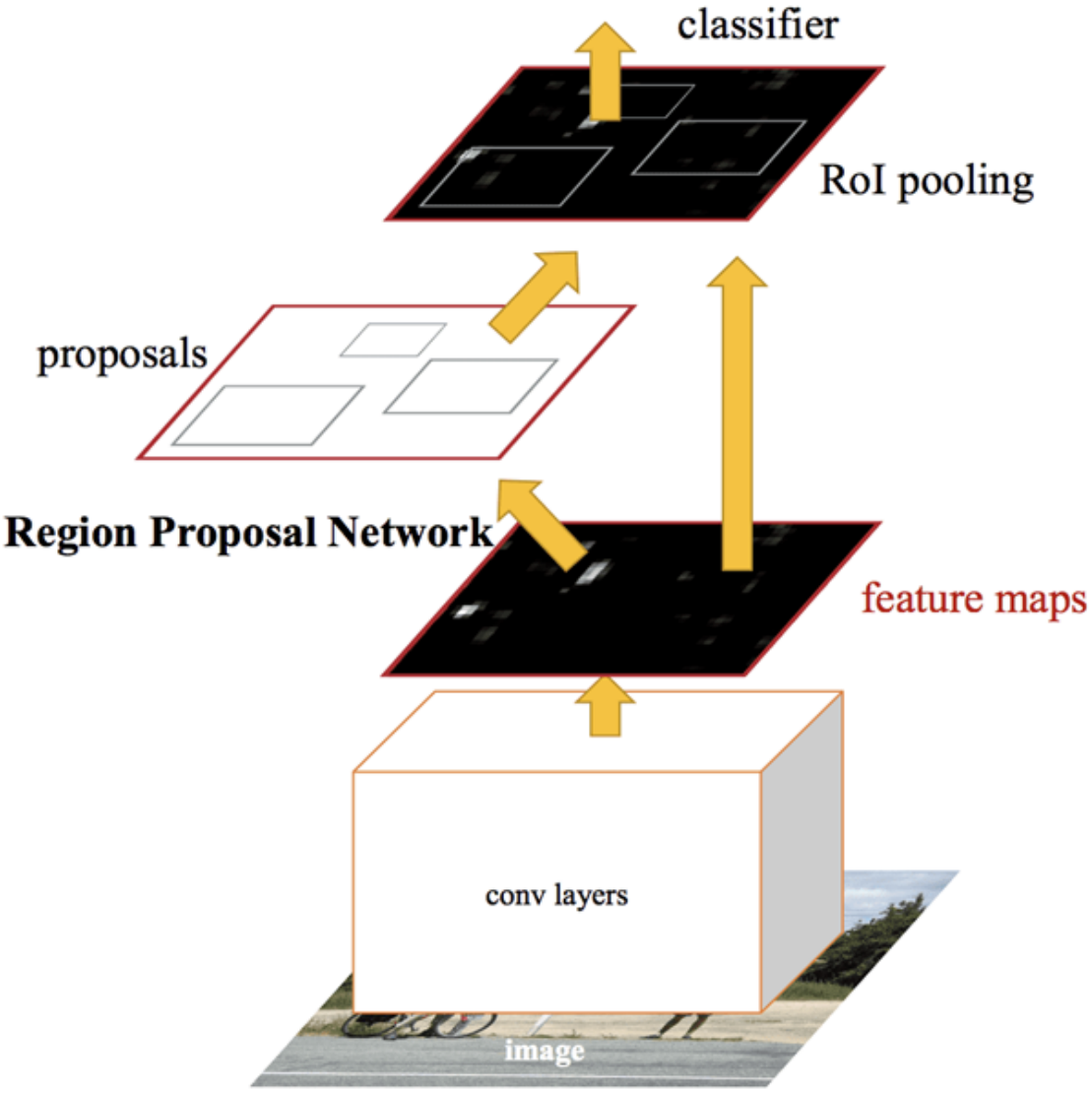

Figure 4.17: Faster R-CNN Model [55]

1D convolutional layers. For the first layer, the kernel size is 2 and the channel number is 128. Similar to the last layer on image input and mask input, the last layer on class-score and ROI input path also flattens the feature map to a tensor of size 32. Different from the other model, the concatenated tensor has a size of 128. Since the number of objects in the image is not fixed, the inputs are truncated or replenished to 10 channels. Figure 4.20 shows the architecture of this proposed model.

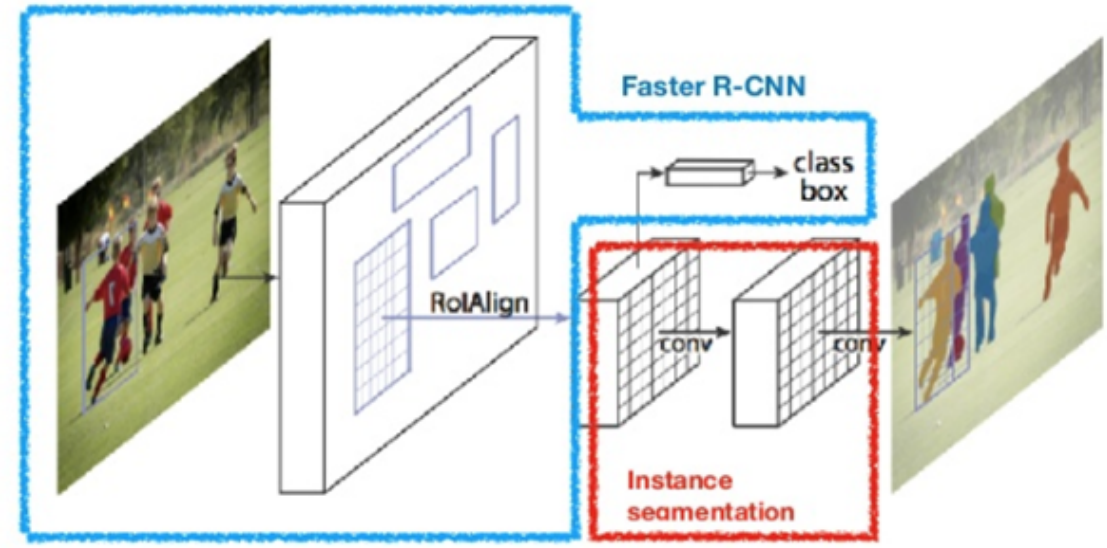

Figure 4.18: Mask R-CNN Model [25]

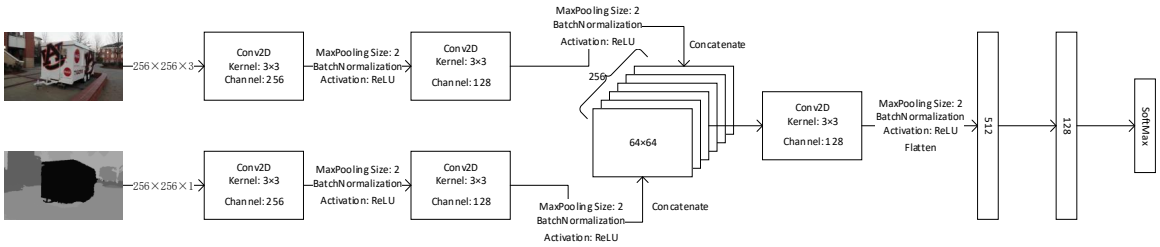

Figure 4.19: Proposed Semantic Segmentation based Model

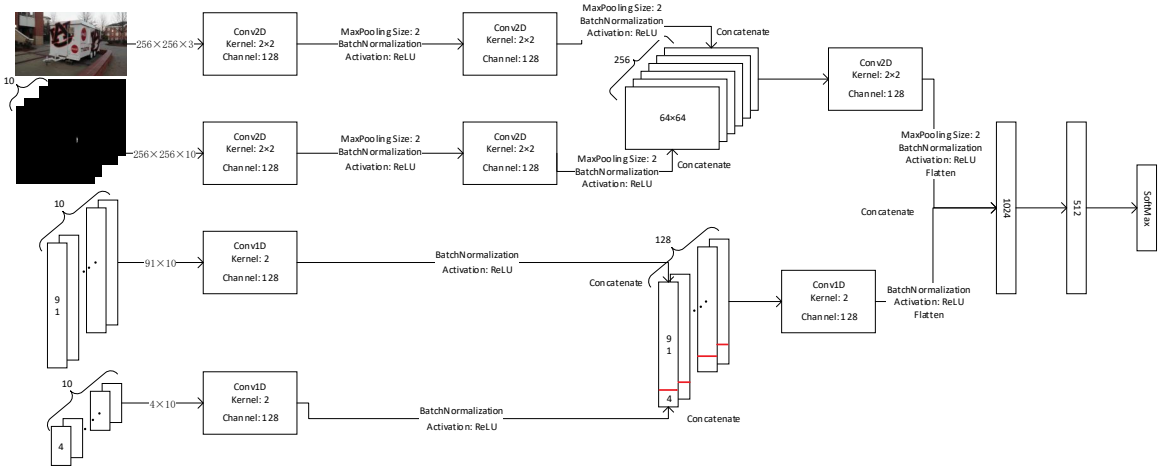

Figure 4.20: Proposed Instance Segmentation based Model

Chapter 5

Experiment Design

Once a satisfying model has been created, it should be installed on the UWP application to navigate the UAV, flying around the obstacles and landing at the desired destination. In this experiment, there are several systems collaborating, including path setting before flight, obstacle avoidance maneuvering, and corresponding path planning. The flow chart 5.1 presents the structure of the whole system.

At the beginning of the experiment, a set of waypoints should be input as an initial course for UAV flight in the application 'SetWaypoints' page. The reasons for not inputting only a destination, but a set of waypoints instead, is that DJI drones don't allow waypoints to be far away from each other, and a course must keep the UAV away from crowds and FAA restricted areas.

Avoiding FAA restricted areas is critical for flying a drone legally. In FAA airspace control rules, airspace is classified with 5 different classes, A, B, C, D, and E, as shown in fig 5.2 [46]. The FAA defines the lower limit of 18,000 feet mean sea level (MSL) up to and including flight level (FL) 600 as Class A airspace. Class B, Class C, and Class D are similar. class B airspace is the airspace that from the ground surface to 10,000 feet MSL surrounding the nation's busiest airports. Class C is from the airport surface to 4,000 feet above the airport elevation around those airports that have an operational control tower, are serviced by a radar approach control, and have a certain number of instrument flight rules (IFR) operations or passenger enplanements. Class D is the airspace from the surface to 2,500 feet above the airport elevation surrounding those airports that have an operational control tower. The definition of these classes airspace areas are individually customized; however, class B consists of a surface

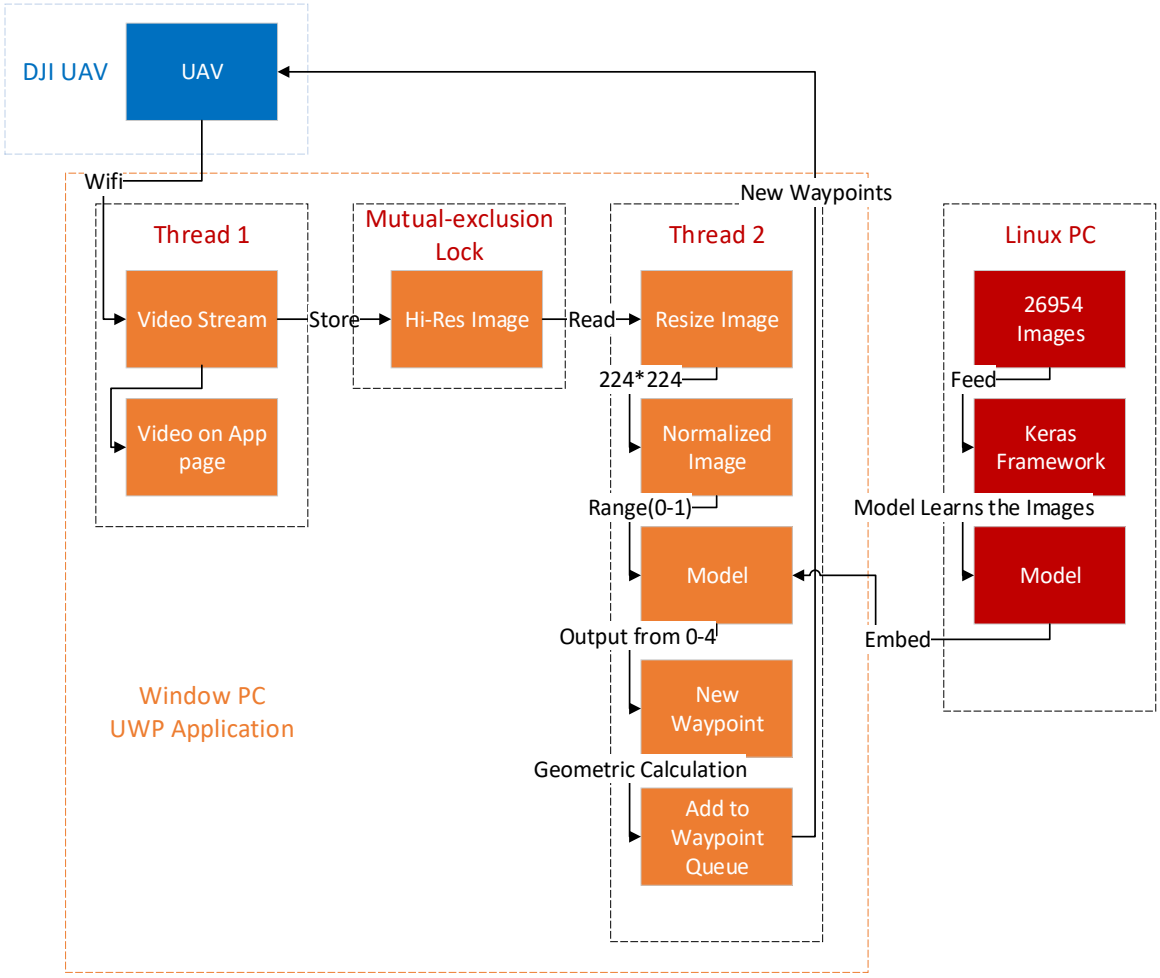

Figure 5.1: System Flow Chart

area and multiple layers, and is designed to contain all published instrument procedures once an unexpected aircraft enters the airspace, class C is usually formed of a surface area with a five nautical mile (NM) radius, an outer circle with a ten NM radius that extends from 1,200 feet to 4,000 feet above the airport elevation, which looks like an upside-down wedding cake and class D have only one layer. Class E airspace is the controlled airspace not classified as Class A, B, C, or D airspace. A large amount of the airspace over the United States is designated as Class E airspace. In most cases, a remote pilot will not need waiver authorization to operate in Class E airspace. Uncontrolled airspace or Class G airspace is the portion of the airspace that has not been designated as Class A, B, C, D, or E. The FAA defines the upper limit at which a pilot can fly a UAV at 400 feet above ground level (AGL); thus, without a waiver, a UAV can only fly at class E and class G. [46]

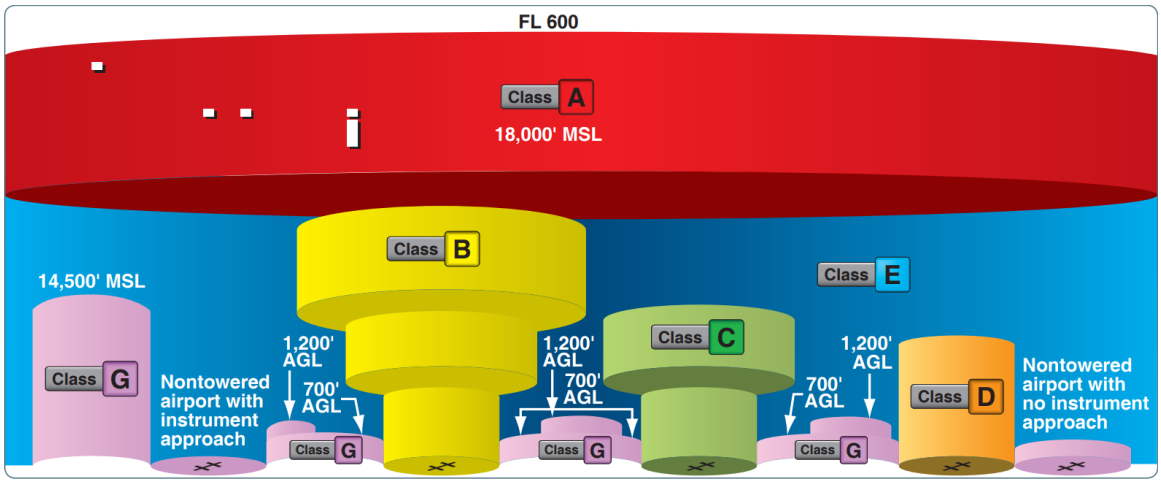

Figure 5.2: Airspace Profile

During a flight in which a UAV is following the preset course, the UAV keeps sending the video stream to the ground station application, and it is made as snapshots stored in a local variable. In the case of memory overflow, a multi-thread structure is implemented on the 'FPV page'. The video stream is working on the original thread, and the neural network model is work on another thread. Since both threads are accessing the image variable come from the video snapshot, a mutual-exclusion lock is applied assuring the resource is never written and read simultaneously, otherwise, an input image of the model could be distorted and affect the precision of the model result.

The path planning part gives different new waypoints according to the five outputs from the deep learning model. For 'keep going' output, there is no new command produced; for 'turn left' and 'turn right', a one-yard away new waypoint on left or right that is perpendicular and on the same level to the current course is added to course; similarly, 'go back' and 'go up' output yields a one-yard away new waypoint that is on the back or above of the drone. As shown in figure 5.3, then the drone sees an obstacle, the path planning would give it a new waypoint, such as the blue dots on the axes. The average collision rate and waypoints completion time will be evaluated in several places around the Auburn University area.

Once the neural network model outputs a command that is not 'KeepGoing', the thread will do the following operation to make sure a new waypoint is inserted into the waypoint queue on the UAV. First, the thread sends a 'Stop Mission' request to the UAV, so the UAV clear its

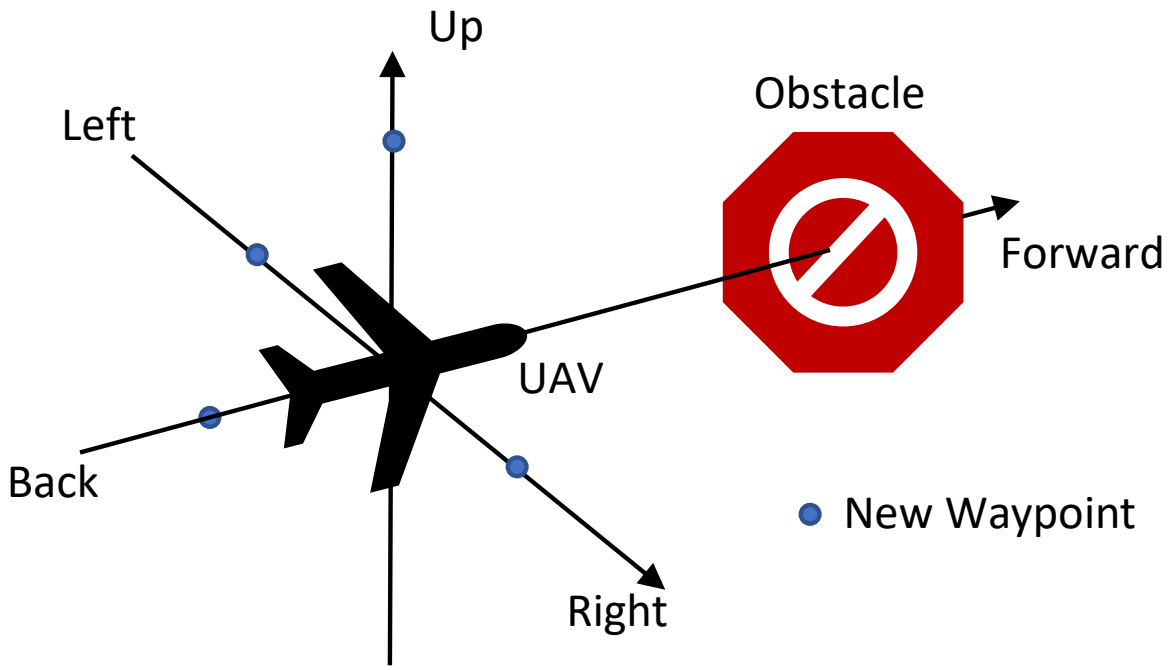

Figure 5.3: Path Planning Mechanism

cache and hover in the air. The second step is inserting the new waypoints to the beginning of the local waypoint queue. The local waypoints mission has to be maintained identical the one on the UAV. Next, we generate a new mission locally and upload it to the UAV. The final step is to start the new mission.

The above step seems to work fine to meet the request. However, during the field flight, the model is constantly generating commands at a high frequency. Theoretically, the frequency should be higher because higher frequency means the model is playing a part efficiently in the obstacle avoidance test. But in fact, sending requests to the UAV at a high frequency will cause a 'Dead Lock' to the system, because every request is only effective when the UAV is on a certain status. For instance, when the UAV is starting a new mission, but the status of the UAV is still uploading the new mission. Since this would make the UAV stop the test, a one second manual pause on the thread is applied.

Even though the manual pause is implemented on the thread to increase the interval between commands, there are still many new waypoints which are densely clustered. Also, when there is a waypoint that is impossible to reach, such as when the waypoint is close to a building or inside a building, the path planning needs a mechanism to escape from this dilemma. As

shown in figure 5.4, when the current location is in a 6 ft. diameter circle of the next waypoint in the waypoint queue, or say the distance between them is lower than 6 ft., the other waypoints in the circle is removed in the circle. This mechanism makes sure the UAV doesn't stay around a location for too much time and also prevents the UAV from being trapped in going anywhere in the building. There is a trade-off between the waypoint accuracy and the obstacle avoidance in this mechanism because the higher the diameter of the circle is the lower the waypoint accuracy is, vice versa.

One obvious problem occurs when the UAV crashes into an obstacle. It ends the test flight and most importantly, damages the expensive drone. The solution to this problem is to make the UAV never collide with an obstacle. How can the obstacle avoidance algorithm be evaluated without actual collision? To determine whether the collision happens, check the distance between the obstacle and the UAV that collected from the front-faced infrared sensors and also check the waypoint mission execution status. A collision is recorded if the distance is shorter than 1 meter (3.28 ft.) and the mission execution status is 'Moving'. In this way, the collision can be determined and also avoided.

During the test flight, there is no way to manually record every event and its attributes; however, they are important for evaluating the test. Such as recording the time and coordinates of waypoints inserting and removing operations will help to check the correctness and efficiency of commands. So, a log to record the details of the test flight shall be made in the application.

The test flight should be performed at a place and a time that has no people or moving cars around in case of an unexpected accident, and the place should have a proper obstacle to challenge the algorithm. The parking lot on 560 Devall Dr., Auburn, Alabama, the U.S. as shown in map figure 5.5, is selected as the test place because there are rare people and cars around on the weekend, and luckily, there is a median of trees that can be a perfect obstacle for the test. Because the limitation of the test flight is 100 meters according to the UAV preset, the test flight is simplified to scenario tests instead of a long-distance flight. The tests require the UAV to fly over the trees or bypass them without any collision. As shown in figure 5.6.

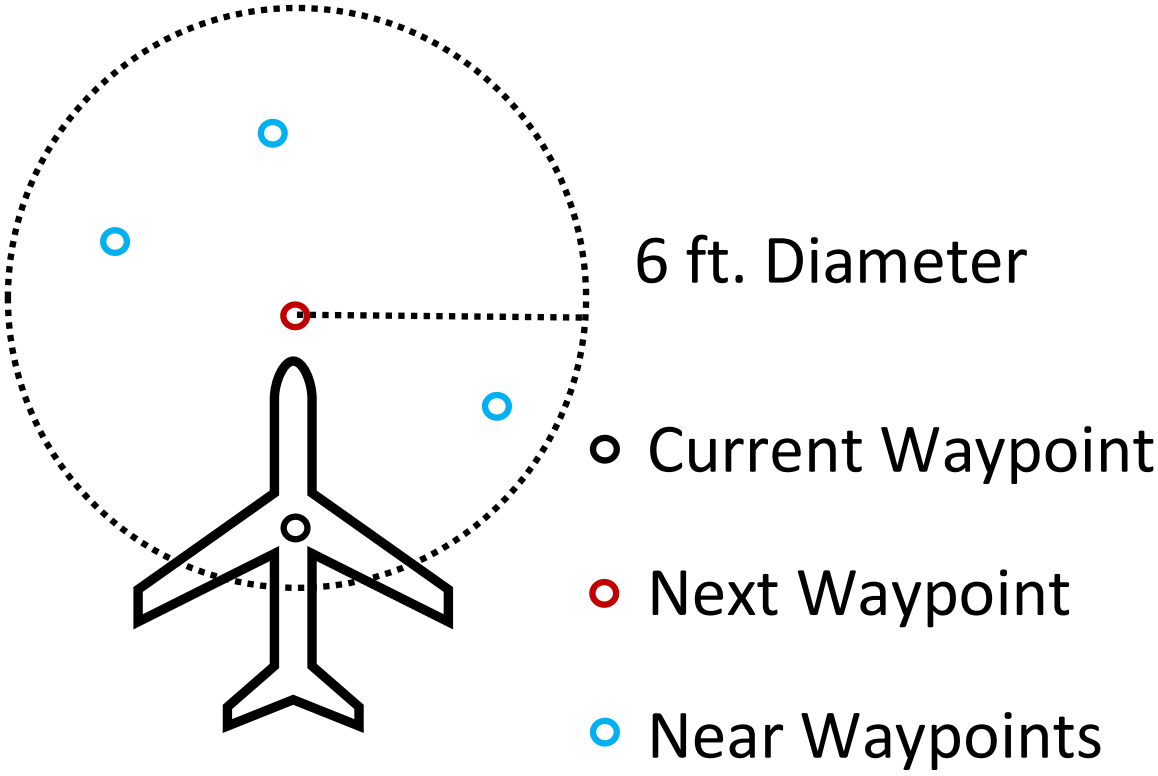

Figure 5.4: Waypoint Pruning Mechanism

Figure 5.5: 560 Devall Dr., Auburn, Alabama

Figure 5.6: The Obstacles in the Field

Chapter 6

Experiment Result

6.1 Application Development

The application pages are designed as in figure 3.4 and figure 3.3. On the top of the FPV page with a drone real-time video stream in figure 3.4, 'Latitude' and 'Longitude' sections show the UAV's current location, and three velocities section indicates the UAV's velocities along the three-dimensional axes. The 'Model Feedback' section has the information of the model responding time during a single-image interference and the output of it.

Figure 3.3 presents the 'Set Waypoints' page. This page has a global map on the right side of the page and it tells where the mission waypoints and the UAV current location are in the real world. On the left section, there are several subsections for the preparation of a mission. 'Set GroundStation Enabled' button actives a function that enables the ground station mode, and this is the first step to do before any other button click. 'Add Waypoint' section has three text boxes that grant users an interface to preset the mission waypoints, which follows the latitude, longitude and altitude order. After a waypoint parameter is input in the three boxes, the 'Add Waypoint' button sends the waypoint into the back of a queue; so the next waypoint can be input. For the following sections, such as 'Create Mission', 'Upload Mission To Aircraft', 'Start Mission', these are all buttons to communicate with the drone for setup. There is one special button: 'Auto Landing' is designed to be a kill switch in case of emergency; because it grants users the authority to make the drone stop moving and land immediately.

6.2 ResNet Family Models Development Result

For the model development part, the results are not satisfying. At the beginning of this project, the selected models are those in the ResNet family, including ResNet50, ResNet101, and ResNet152, because of their excellent performance on the image classification problems. To transfer the ResNet model to this project, the last layer, which is a 1000 nodes softmax layer, is replaced by a 5 nodes softmax layer. Except the last layer, the other layers keep the weight that is trained with ImageNet. With a DJI drone that was acquired for shooting videos, there were 26,954 snapshot pictures used for training and testing image sets for model development.

6.2.1 ResNet152

ResNet152 is a model with very deep architecture is focusing on solving tasks that have complicated logic or high dimensions. As shown in figure 6.1, the training accuracy is almost 99%, the validation accuracy is around 45%, and meanwhile, the validation loss increases as the training accuracy converges. The overfitting problem exists no matter what regularization techniques are applied. The techniques include data normalization, data augmentation, and adding regularization terms. The hyperparameters for this model are batch size 8, epoch number 50, l1 regularization 0.1, and l2 regularization 0.1.

6.2.2 ResNet101

Given the result of ResNet152 has an overfitting problem, the Resnet101 model is tested. According to the nature of deep learning, a deeper network usually makes model variance higher, which solves the underfitting problem but exacerbates the overfitting problem. As shown in figure 6.2, similar to the result of ResNet50, the training accuracy is around 95%, the validation accuracy is below 40%. As for the losses, the training loss converges very well, but the validation loss continuously increases. The result is produced under the following training hyperparameters: batch size 16, epoch number 80, l1 regularization 0.1, and l2 regularization 0.1. Despite this test is under instruction by the nature of deep learning, the overfitting problem is still preserved.

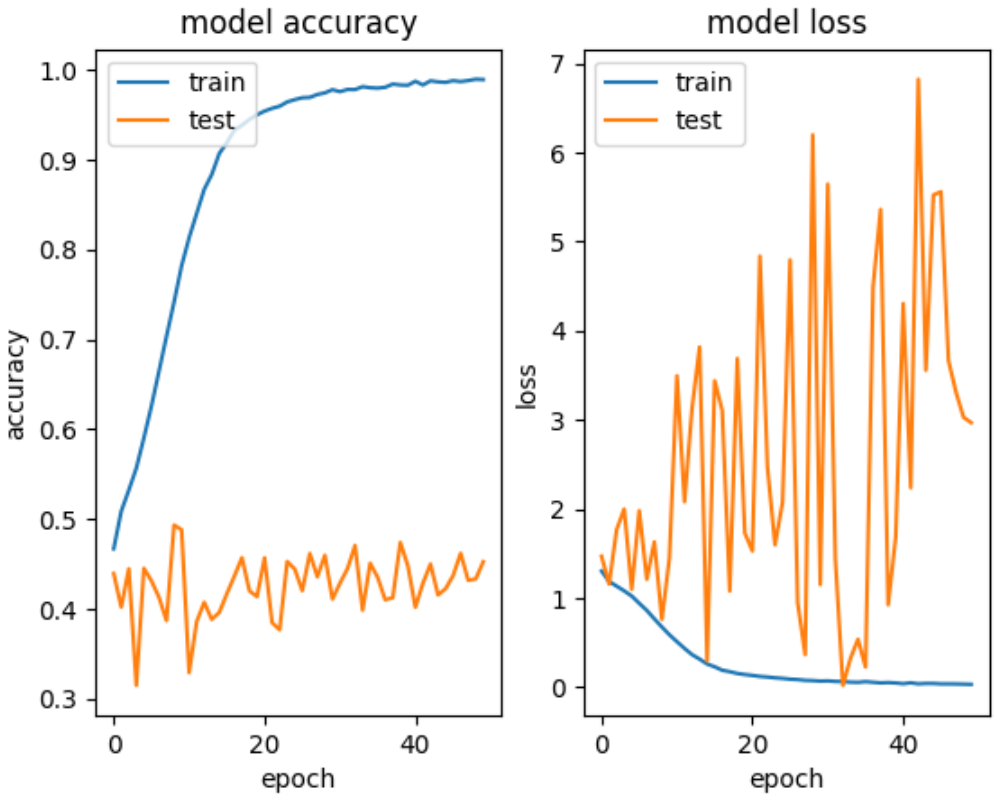

Figure 6.1: Model Training Plot of ResNet152

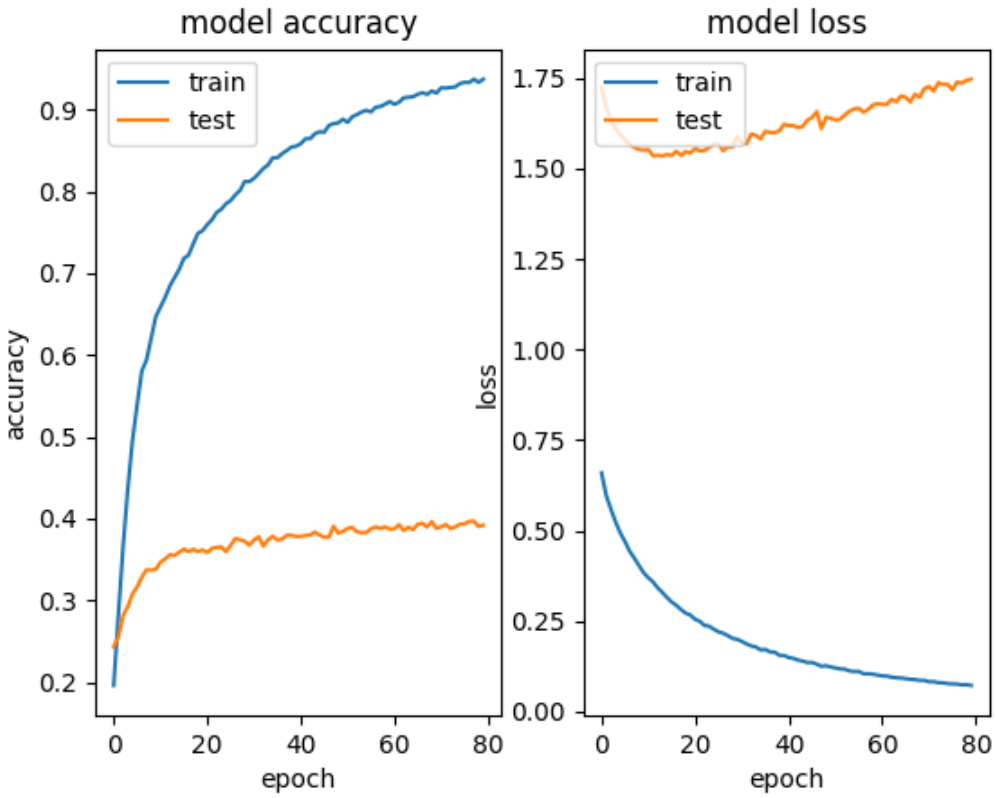

Figure 6.2: Model Training Plot of ResNet101

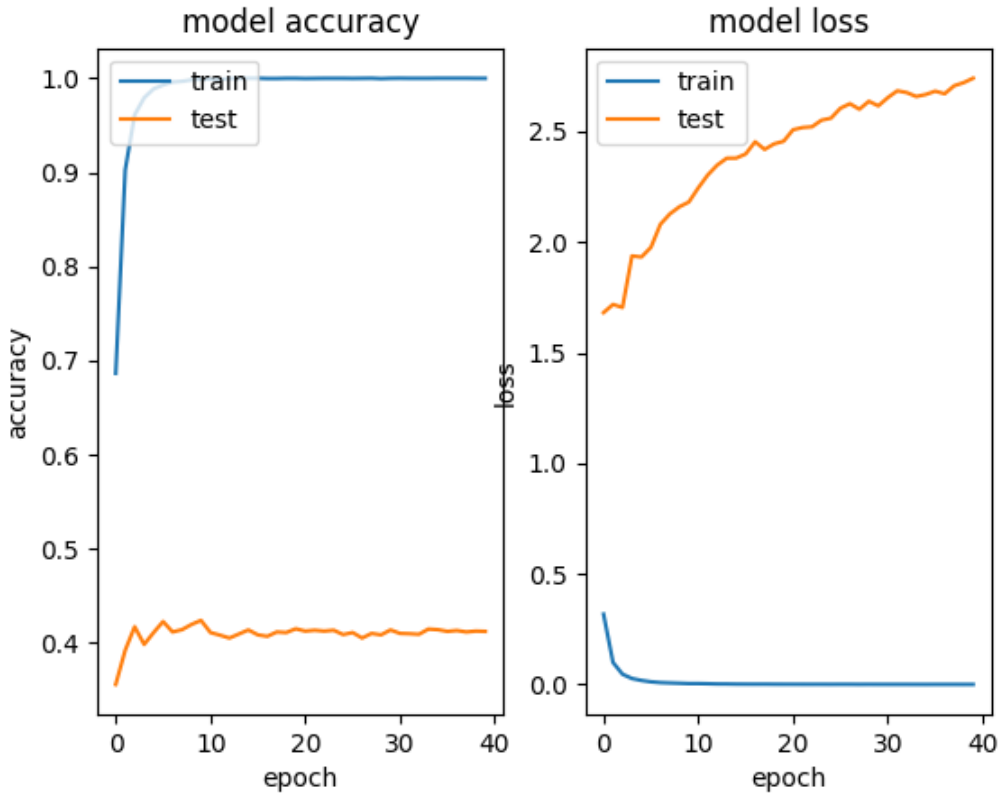

Figure 6.3: Model Training Plot of ResNet50

6.2.3 ResNet50

The last member in the ResNet family is called ResNet50. Since the ResNet152 and ResNet101 both have high variance, ResNet50 is promising in solving this problem because of its shallower layers. As shown in figure 6.3, the training accuracy is almost 100%, the validation accuracy is over 40%. Similar to the losses of the above models, training loss converges, and the validation loss diverges. The training is carried out with the following hyperparameters, batch size 32, epoch number 40, l1 regularization 0.1, and l2 regularization 0.1. Unfortunately, the models in the ResNet family are all tested and proved to be unsuitable for the project. Every model has an overfitting problem.

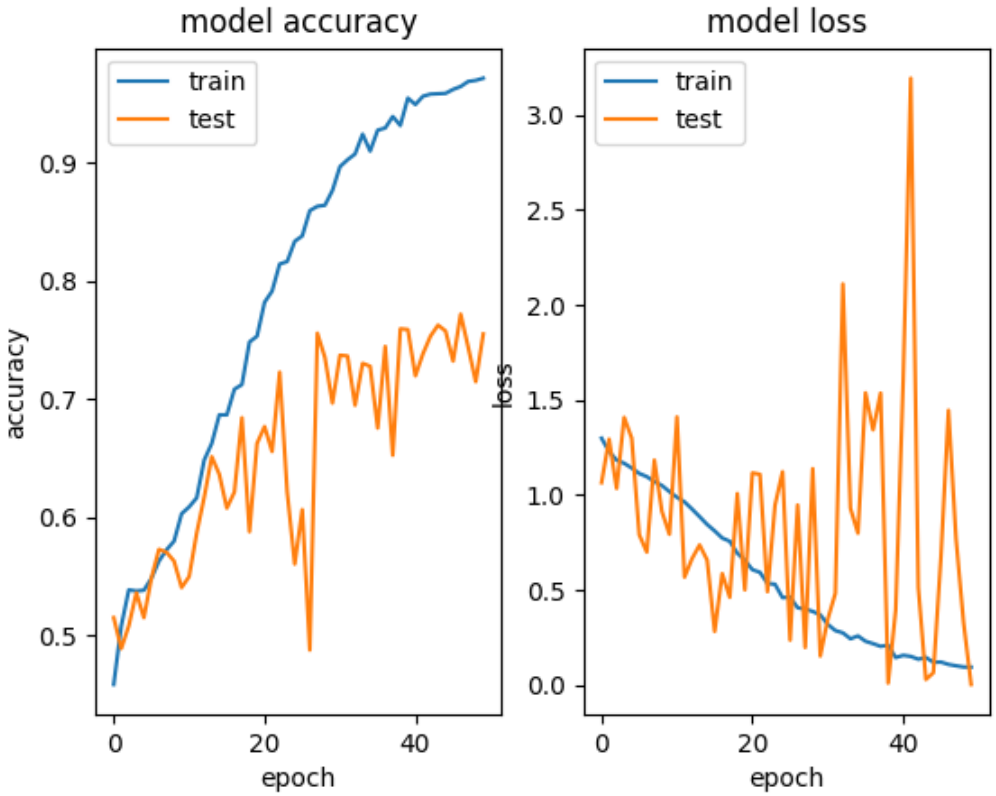

Figure 6.4: Model Training Plot of ResNet152 based GRU Model

6.3 RNN Family Models Development Result

6.3.1 ResNet152 based GRU Model

Figure 6.4 show the result of the model applying GRU on ResNet152. From the figure, one obvious thing is that the model still has the overfitting problem, which means the RNN doesn't solve the problem completely. The training accuracy has been improved from 40% to over 70%, which seems to be a great improvement. However, the accuracy is fluctuating, and it means that during the validating stage, a portion of the validation data set is classified randomly, as the correct predict number fluctuating. This is another behavior of overfitting problem. This can also be proved from the loss chart, which is largely fluctuating. The hyperparameters adopted are batch size 4, frame number (the number of sampled consecutive images) 5, latent dimension 256.

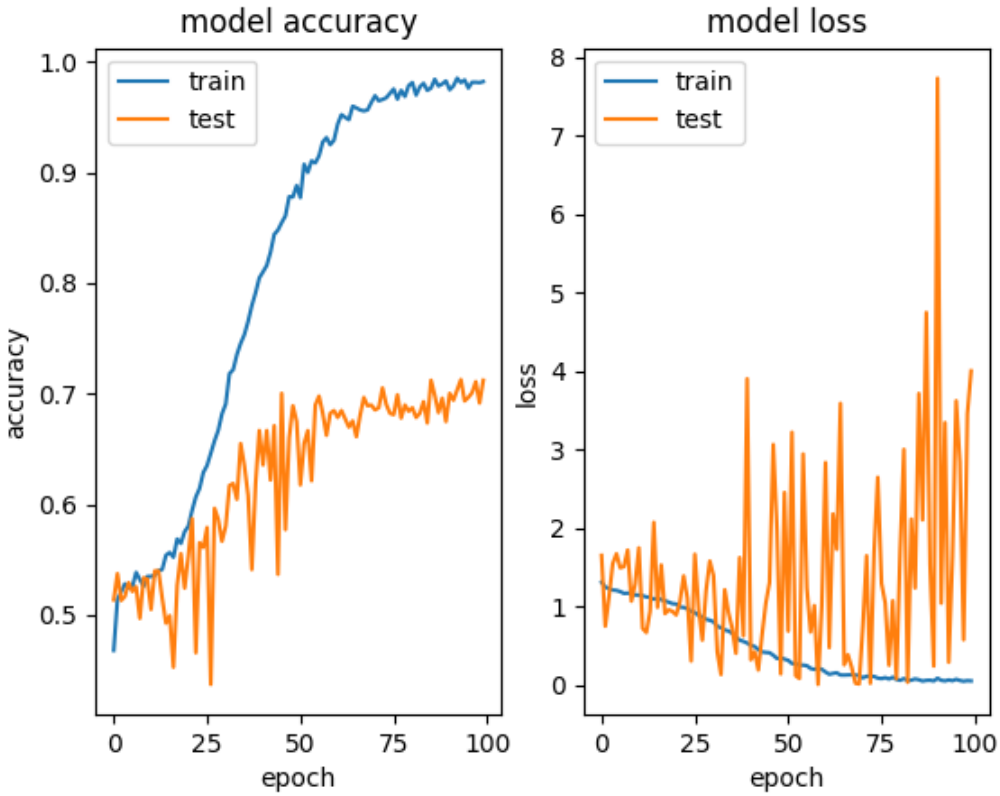

Figure 6.5: Model Training Plot of ResNet101 based GRU Model

6.3.2 ResNet101 based GRU Model

The result of the GRU applied on ResNet101 are shown in figure 6.5. Compared to the result of ResNet101, this chart shows that overfitting remains, but it is mitigated. However, after trying various methods to reduce the gap between training and validation accuracy, the best validation accuracy performance is fluctuating around 75%, and the cost is also fluctuating in a large range, which indicates the model is overfitted. The hyperparameter in this model is batch size 4, frame number 5, latent dimension 256.

6.3.3 ResNet50 based GRU Model

The last RNN model applies GRU on ResNet50. As shown in figure 6.6, different from those models containing ResNet101 and ResNet152, this model has a strange behavior. The training accuracy is around 75% and the validation accuracy is fluctuating around 63%, which are not close to each other and both very low. This is a behavior of both underfitting problems and

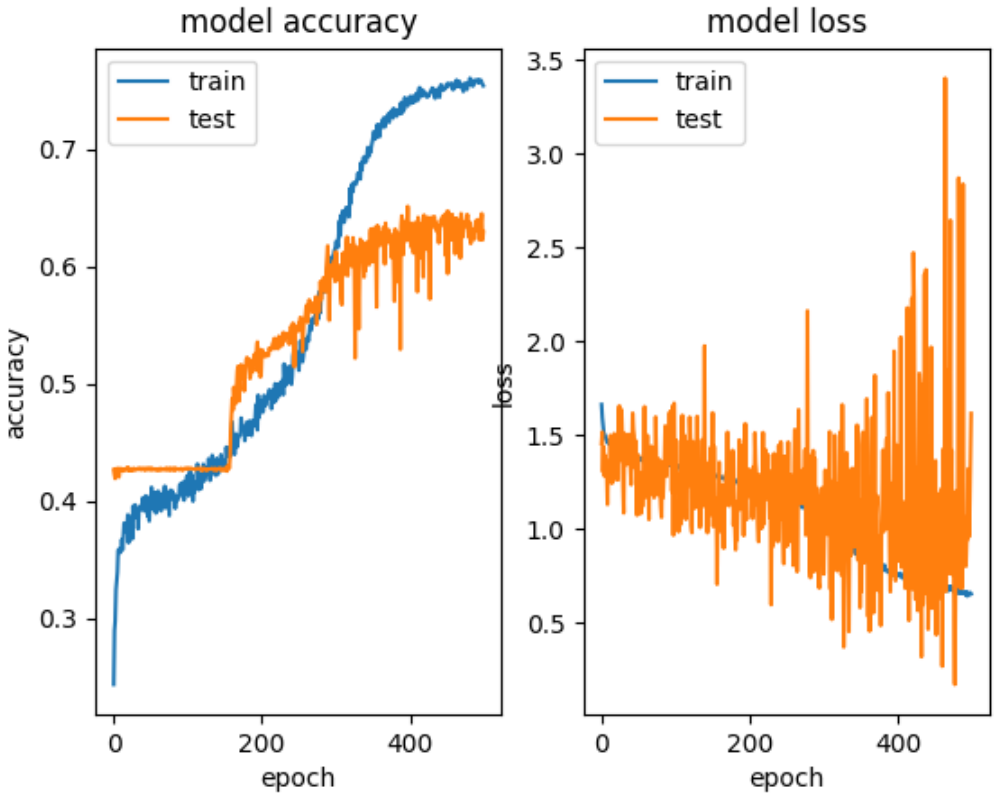

Figure 6.6: Model Training Plot of ResNet50 based GRU Model

overfitting problems. The reason why the model has both problems is that the dataset size shrinks frame number times and the architecture is halved, thus the model is incapable to do the job meanwhile it cannot find the correct relation between features.

6.4 ResNet Family Models Development Result Using GANs Data Augmentation

One of the effective methods to solve an overfitting problem is to increase the size of the dataset, and this trick reduces the overfitting effect but doesn't bring an underfitting problem. The size of the dataset for the experiments above is 21400 images in the training set and 5554 images in the test set, thus 26,954 images in total. After applying GauGANs and screening out those illegible synthetic images, the size of the dataset is increased to 26568 training images and 6642 test images, thus 33210 images in total. There are over 6000 images are synthetically generated to fix the overfitting problem.

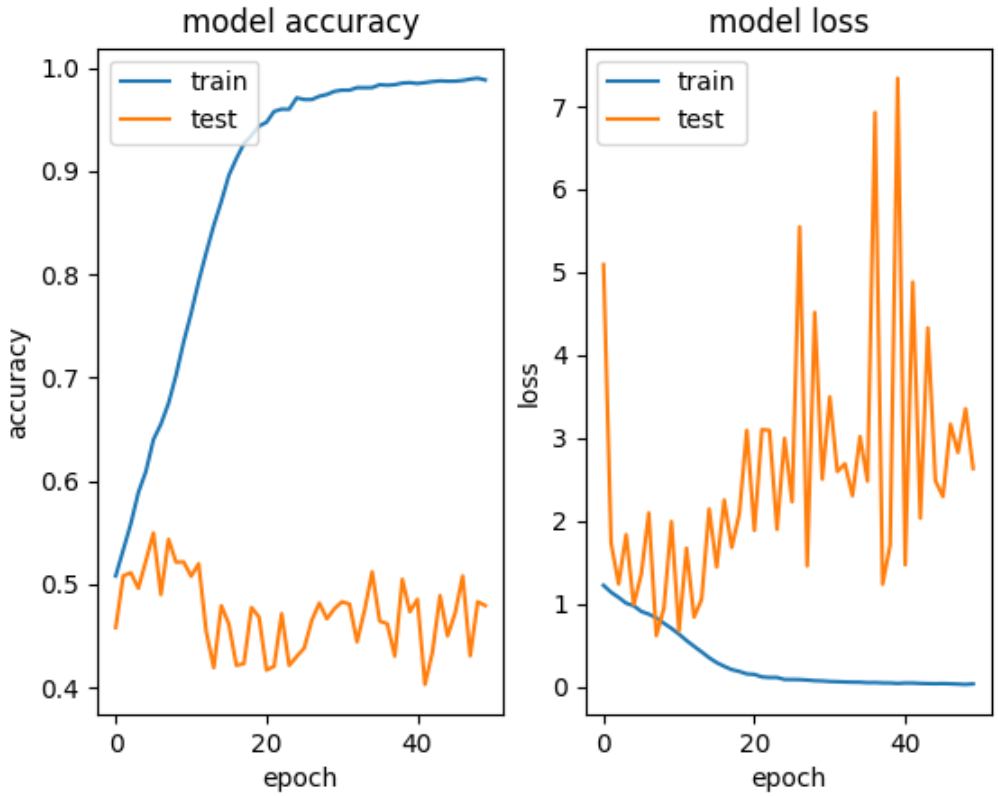

Figure 6.7: Model Training Plot of ResNet152 Using Synthetic Images

6.4.1 ResNet152 Using Synthetic Images Data Augmentation

Figure 6.7 shows the result after applying data augmentation which increases the dataset size by adding synthetic images generated by GauGAN. Compared to the result before adding the synthetic images, the figure shows that the accuracy is slightly increased. On the other hand, the accuracy is fluctuating. The reason is that even though the dataset size is increased by over 6000, compares to the based size of 26000, it's still not big enough to turn it around. Thus, from the accuracy and the loss charts, it is safe to say the model is still overfitting.

6.4.2 ResNet101 Using Synthetic Images Data Augmentation

Compared to the result of ResNet152, the overfitting problem is mitigated slightly as shown in ResNet152's figure 6.8. The validation accuracy is improved from below 40% to over 50%, even though the cost is increased from around 1.75 to around 2.5. The overfitting problem still preserves after increasing the dataset size.

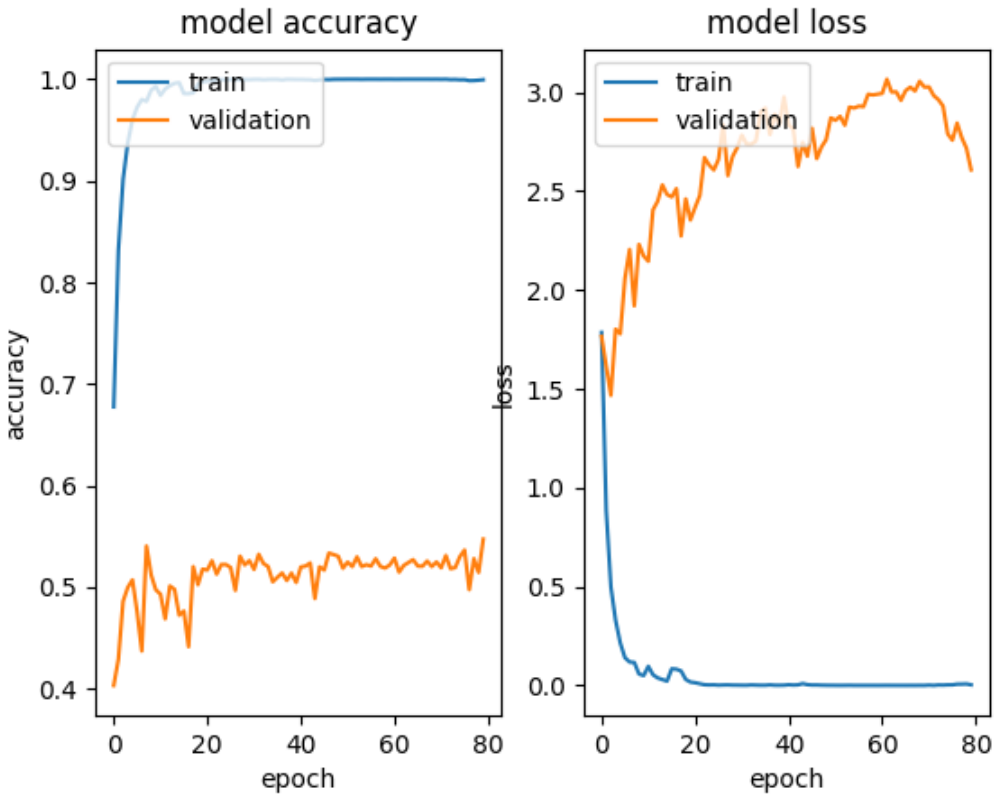

Figure 6.8: Model Training Plot of ResNet101 Using Synthetic Images

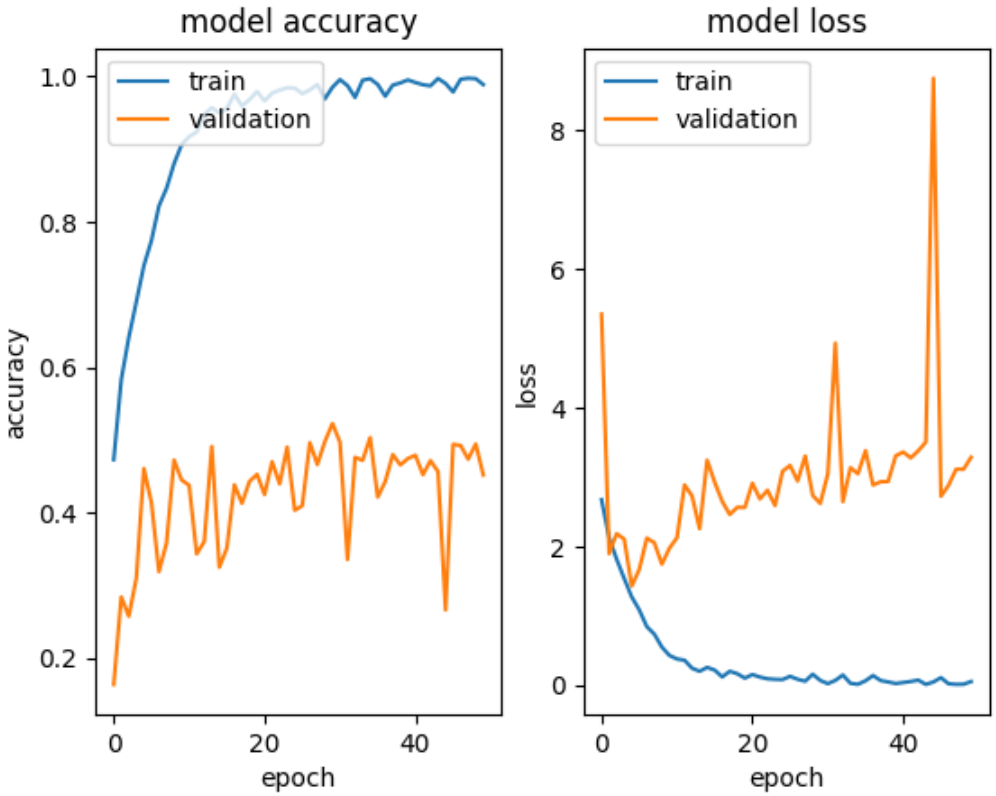

Figure 6.9: Model Training Plot of ResNet50 Using Synthetic Images

6.4.3 ResNet50 Using Synthetic Images Data Augmentation

Similar to the ResNet152 and ResNet101, ResNet50 performs a little better on validation accuracy after applying the data augmentation as shown in figure 6.9. Although the size of the dataset improves the accuracy, the enlargement of divergency creates more randomness to the validation process. The fluctuating wave tells the randomness attribute. After all, the result after adding synthetic images as the data augmentation method improves the accuracy but also makes the accuracy fluctuating. To solve this problem, adding more synthetic images could be a promising technique, however, due to the limit of time and training resources, this project won't explore more on this topic but looking for other feasible techniques.

From the table 6.1, it is clear that E2E models have bad validation accuracy, which means that the models have serious overfitting problems. The RNN models have the best performance on the overfitting problem, however, the fluctuating validation accuracy reveals the randomness in the evaluation, which is also caused by overfitting effect. As mention in [22], an E2E model

usually requires a lot of data, especially when a task is complicated, as the task in this project. It is hard to tell what relations between the input and the output are that the model relied on. Therefore, even though the models are learning very well on the training dataset, they can't predict correctly on the validation dataset, because they use the wrong mapping relations. The metrics in table 6.2 also prove that the E2E models perform poorly because of the overfitting problem since they are close to validation accuracy but away to training accuracy. In the table indexes, there are accuracy, precision, recall and F1 score. The reason why precision, recall and F1 score are adopted is that accuracy can reflect the value of the model when the true positive and true negative is important; on the other hand, the F1 score can reflect the quality of a model when the dataset is imbalanced, and the dataset is normally imbalanced including the dataset in this paper. Because the dataset is imbalanced, the precision, recall and F1 score are measured macro-weighted, to mitigate the misleading of the major number labels.

	ResNet50	ResNet101	ResNet152
Standard	99.84%/41.84%	92.54%/39.15%	98.89%/44.12%
RNN	75.17%/65.89%	98.71%/71.95%	98.28%/75.94%
Data Augmentation	98.72%/46.27%	99.84%/55.48%	98.70%/47.87%

Table 6.1: E2E Models Accuracy (Training Accuracy/Validation Accuracy)

	Accuracy	Precision	Recall	F1 Score
Standard ResNet50	41.49%	41.75%	40.85%	41.29%
Standard ResNet101	42.89%	42.99%	41.94%	42.44%
Standard ResNet152	44.95%	45.08%	44.10%	44.55%
ResNet50 (RNN)	63.50%	49.52%	60.69%	63.38%
ResNet101 (RNN)	62.87%	66.71%	58.77%	61.91%
ResNet152 (RNN)	65.36%	68.20%	61.38%	64.09%
ResNet50 (Data Augmentation)	48.24%	51.13%	48.24%	46.51%
ResNet101 (Data Augmentation)	36.68%	35.56%	36.68%	30.63%
ResNet152 (Data Augmentation)	33.44%	38.61%	33.44%	27.24%

Table 6.2: E2E Models Evaluation

6.5 Semantic or Instance Segmentation based Multi-Stage Model

6.5.1 Semantic Segmentation based Multi-Stage Model

Different from the models above, the proposed semantic segmentation based multi-stage model requires the semantic segmentation model, which is a pretrained DeepLabv2 on COCO Stuff and Thing dataset [3] in this project. It inferences the input image and produces semantic images, and all of the original images and corresponding semantic images are stored separately. During the training, the training framework finds the corresponding inputs for the two model inputs. This way, the model benefits a lot from the profits the DeepLabv2 makes in the COCO Stuff and Thing dataset, even though the proposed model has never seen a large dataset as COCO dataset. Theoretically, the performance would be dramatically improved.

From the figure 6.10 it is clear that the proposed semantic segmentation-based multi-stage model has an obvious improvement compared to the E2E models. The main reason is that instead of letting the model figure out the relation between the inputs and the output, which requires so much data, the multi-stage model has human guidance. In this model, a semantic image is produced for the model from a pretrained model that has seen a lot more data before the project. Thus the proposed model know to find the relations between the labels and the images' semantics. Due to the limitation of data, the model never knows to figure out this logic, unless implementing human interference.

6.5.2 Instance Segmentation based Multi-Stage Model

The output of the instance segmentation model, which is Mask RCNN in this project, is quite different from the output of the semantic segmentation model. The semantic segmentation model generates only one semantic image for each image, however, the instance segmentation model has an uncertain number of outputs. The number of the output is decided by the instance number in the image, so the input of the proposed model followed by the instance segmentation model should be truncated or replenished to a fixed number. Ten channels are adopted in the project. The output of the instance segmentation model has more rich information, because it

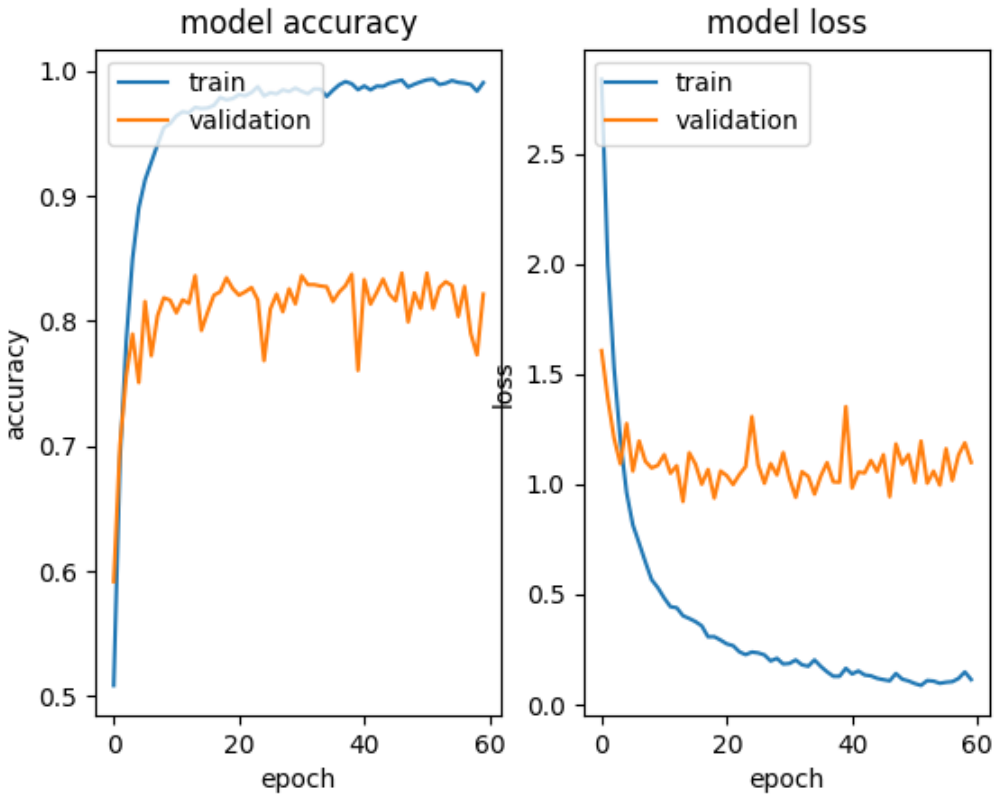

Figure 6.10: Model Training Plot of Semantic Segmentation based Model

not only tells what the objects are, nut it also distinguishes the different instances among the same objects, such as in a crowd of people.

The figure 6.11 shows that the result of the proposed instance segmentation-based multi-stage model has a comparable result with the semantic segmentation-based multi-stage model. As discussed in this paper, the differences between instance segmentation and semantic segmentation are instance segmentation contains more information such as the instances of objects; however, the adopted pretrained instance segmentation, Mask RCNN, is pretrained on the COCO vanilla database, which only has 80 'things labels'. There are several really common seen outdoor objects in the extra 91 'stuff labels', such as trees, roads, grass. This should be the reason why the proposed instance segmentation-based multi-stage model does not outperform the semantic segmentation-based multi-stage model.

The table 6.3 shows the training and validation result of the two models and the table 6.4 shows the test result of the two models. From the tables, it is clear that the metrics of test dataset are close to the validation accuracy, which means the the models perform fine on the imbalanced dataset. One of the reason is that the classes during the training are weighted by the inverse proportion number.

	Semantic based Model	Instance based Model
Training Accuracy	98.51%	98.19%
Validation Accuracy	82.61%	81.71%

Table 6.3: Multi-stage Models Accuracy

	Accuracy	Precision	Recall	F1 Score
Semantic based Model	84.65%	84.79%	84.64%	84.69%
Instance based Model	78.96%	79.14%	78.95%	78.96%

Table 6.4: Multi-stages Models Evaluation

6.6 Test Flight

The test flight is taken at The parking lot on 560 Devall Dr., Auburn, Alabama, the U.S. as demonstrated in the 'Experiment Design' section, the main obstacles are the trees and the light

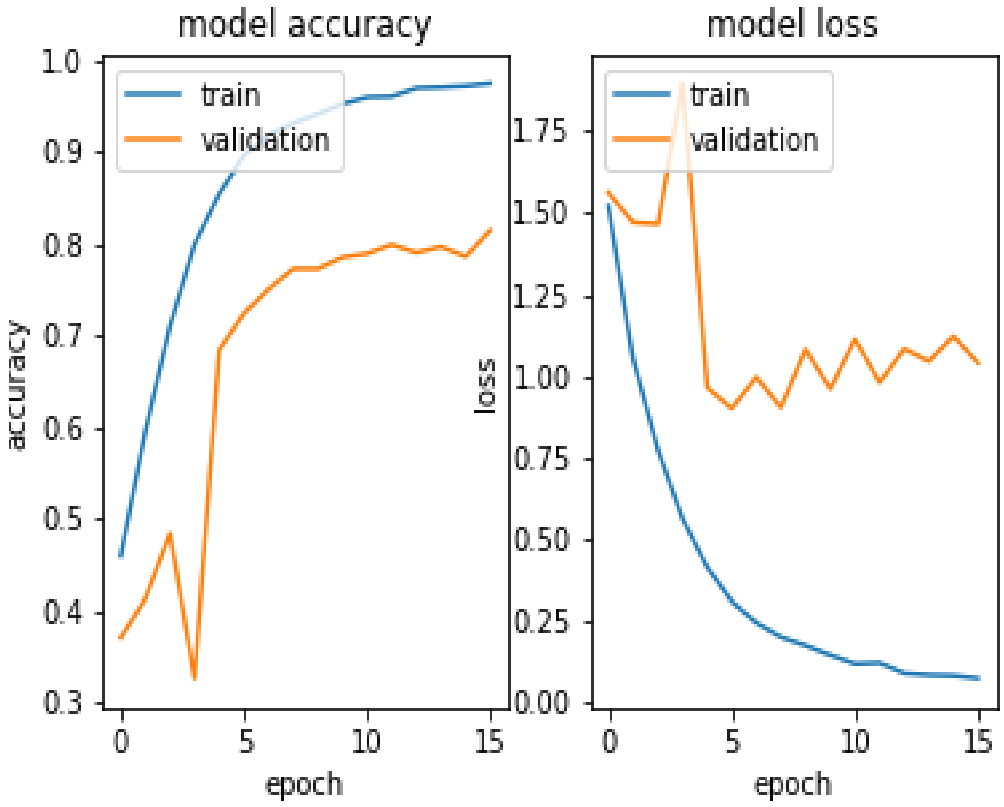

Figure 6.11: Model Training Plot of Instance Segmentation based Model

poles in the median of the parking lot. When the drone takes off, the model starts working straight away. The drone can behave in the ways the training data is labeled, such as backing up when it's too close to the trees and the light poles and keeping going when the drone is above the tree or when there is no obstacle in front of it. Finally, the drone reaches its destination without any collision recorded.

In the test flight, the average model inference time for the proposed semantic segmentation-based model is recorded as 2.719 seconds, which is expensive for a real-time UAV system even though the flight behavior is acceptable. The reason why the inference time is high is that the DeepLab model is huge. It takes a major part of the time cost. Because of the latency between acquiring the image and outputting the command, the behavior of the UAV is lagging. Thus, this model is working better when the flying speed is relatively slow, however, this would harm the efficiency of the mission.

On the other hand, the average model inference time for the proposed instance segmentation-based model is recorded as 1.460 seconds, which is smaller than the proposed semantic segmentation-based model's record. However, it is still not ideal for real-time UAV obstacle avoidance reaction time. The reason the model has a long reaction time is that the parameters are too many in the model. To fix this problem, or say to reduce the size of the model, there will be a risk of an underfitting problem.

Chapter 7

Conclusion

In this research many Deep Learning models are trained and tested for the UAV obstacle avoidance algorithm, including E2E models and multi-stage models. The training result and the validation result shows that E2E models have various levels of overfitting problem, which means the models have difficulty figuring out the relation between labels and input itself. Specifically, the validation accuracy is lower than 50% when the training accuracy is almost 100%, or the validation accuracy is strongly fluctuating. The main reason for the phenomenon is that the data is too limited to make the model understand the correct input-output mapping.

On contrary, the multi-stages models have better performance overfitting problems. The proposed semantic segmentation-based model has the best validation accuracy in all the models that are tested in this paper. Compared to the E2E models, the accuracy is improved by over 30% which reaches 82% accuracy. The other evaluation metric also proves that the model has a good performance even though the dataset is imbalanced. As for the proposed instance segmentation-based model, the accuracy is around 80% which is comparable to the 82% semantic segmentation-based model accuracy.

The test flight shows that the semantic-segmentation based model has latency between the image acquisition time and the model output time. This shows that the model is suitable for UAV obstacle avoidance when the flying speed is relatively slow; otherwise, there is a safety concern. The model inference time plays a big part in the real-time obstacle avoidance algorithm, the shorter it is the faster the UAV can react. To reduce the number of parameters, meanwhile keeps the model quality, more time to tune the model is required to get rid of the underfitting risk.

Chapter 8

Future Work

The reason why the E2E models in this paper have an overfitting problem is that the model can't find the relation between input and output due to the limitation of training data. Capturing more data is worthy to try to mitigate the overfitting problem, after all getting more data is one of the methods to lower the variance without increasing the bias.

For RNN models, one thing worthy to try is changing the input of RNN structure to semantic or instance maps. Since adopting multi-stage models can improve the base CNNs' models performance, it may improve RNN performance as well. For example, as shown in the figure 4.5, the ResNet's output feature map is input to the GRU unit, the input can be changed to semantic or instance maps.

In the data augmentation part, the adopted GauGAN did a great job of synthesizing new images. However, the number of generated images is not ideal because every GauGAN model can only generate one synthetic image with an original image. To gain more synthetic images, training more GauGAN models may be a feasible way to do.

Like what has been discussed in previous sections, the adopted Mask RCNN model is a pretrained model that is trained on COCO 'things' dataset. To improve the performance of the instance segmentation-based model, maybe improve the Mask RCNN model's performance is a choice. However, training a Mask RCNN model on COCO 'stuff-things' dataset is not an easy task, because it doesn't provide required the information, such as ROIs, Class IDs, and mask maps. Thus, figure out a way to generate the required data is the key.

References

[1] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[2] H. Caesar, J. Uijlings, and V. Ferrari, "Coco-stuff: Thing and stuff classes in context," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1209–1218, 2018.

[3] H. Caesar, J. R. R. Uijlings, and V. Ferrari, "Coco-stuff: Thing and stuff classes in context," *CoRR*, vol. abs/1612.03716, 2016.

[4] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, and P. Campoy, "A review of deep learning methods and applications for unmanned aerial vehicles," *Journal of Sensors*, vol. 2017, 2017.

[5] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. V. Eycken, "Cnn-based single image obstacle avoidance on a quadrotor," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2759–2764, May 2017.

[6] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[7] L.-C. Chen and Y. Zhu, "Semantic image segmentation with deeplab in tensorflow." `https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html`. Accessed: 2021-02-24.

[8] P.-H. Chen and C.-Y. Lee, "Uavnet: An efficient obstacel detection model for uav with autonomous flight," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, pp. 217–220, IEEE, 2018.

[9] Q. Chen and V. Koltun, "Photographic image synthesis with cascaded refinement networks," *CoRR*, vol. abs/1707.09405, 2017.

[10] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016.

[12] F. D, "Batch normalization in neural networks." `https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c`, Oct 2017. Accessed: 2019-10-04.

[13] F. David and W. Li, "Convert tf.keras/keras models to onnx." `https://github.com/onnx/keras-onnx`.

[14] DJI Support, *Mavic Air User Manual*. SZ DJI Technology Co., Ltd.

[15] F. H. K. dos Santos Tanaka and C. Aranha, "Data augmentation using gans," *CoRR*, vol. abs/1904.09135, 2019.

[16] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, pp. 2366–2374, 2014.

[17] C. Eliot, "Convert ml models to onnx with winmltools." `https://docs.microsoft.com/en-us/windows/ai/windows-ml/convert-model-winmltools`. Accessed: 2020-01-02.

[18] C. Eliot and Q. Radich, "Windows machine learning." https://docs.microsoft.com/en-us/windows/ai/windows-ml/. Accessed: 2020-01-01.

[19] F. Fedorenko and S. Usilin, "Real-time object-to-features vectorisation via siamese neural networks," in *Ninth International Conference on Machine Vision (ICMV 2016)*, vol. 10341, p. 103411R, International Society for Optics and Photonics, 2017.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[21] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[22] T. Glasmachers, "Limits of end-to-end learning," *CoRR*, vol. abs/1704.08305, 2017.

[23] U. GUPTA, "Detailed guide to understand and implement resnets." https://cv-tricks.com/keras/understand-implement-resnets/. Accessed: 2019-10-03.

[24] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: State of the art," *CoRR*, vol. abs/2007.00047, 2020.

[25] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[27] M. Hua, Y. Nan, and S. Lian, "Small obstacle avoidance based on RGB-D semantic segmentation," *CoRR*, vol. abs/1908.11675, 2019.

[28] X. Huang, Y. Li, O. Poursaeed, J. E. Hopcroft, and S. J. Belongie, "Stacked generative adversarial networks," *CoRR*, vol. abs/1612.04357, 2016.

[29] J. Hui, "Real-time object detection with yolo, yolov2 and now yolov3." https://medium.com/@jonathan_hui/

real-time-object-detection-with-yolo-yolov2-28b1b93e2088. Accessed: 2019-10-04.

[30] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.07004, 2016.

[31] K. Jacobs, "Python deep learning tutorial: Create a gru (rnn) in tensorflow." https://www.data-blogger.com/2017/08/27/gru-implementation-tensorflow/. Accessed: 2021-02-26.

[32] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "Rednet: Residual encoder-decoder network for indoor RGB-D semantic segmentation," *CoRR*, vol. abs/1806.01054, 2018.

[33] V. K. Jonnalagadda, "Object detection yolo v1 , v2, v3." https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a, Jan 2019. Accessed: 2019-10-04.

[34] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018.

[35] S. Konam, "Vision-based navigation and deep-learning explanation for autonomy," *Ph. D. thesis, Masters thesis*, 2017.

[36] C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris, and C.-S. Bouganis, "Dronet: Efficient convolutional neural network detector for real-time uav applications," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 967–972, IEEE, 2018.

[37] H. Y. Lee, H. W. Ho, and Y. Zhou, "Deep learning-based monocular obstacle avoidance for unmanned aerial vehicle navigation in tree plantations," *Journal of Intelligent & Robotic Systems*, vol. 101, no. 1, 2020.

[38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[39] D. Matthew and R. Fergus, "Visualizing and understanding convolutional neural networks," in *Proceedings of the 13th European Conference Computer Vision and Pattern Recognition, Zurich, Switzerland*, pp. 6–12, 2014.

[40] J. Minguez, L. Montano, and O. Khatib, "Reactive collision avoidance for navigation with dynamic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 588–594, IEEE, 2002.

[41] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014.

[42] R. Miyamoto, Y. Nakamura, M. Adachi, T. Nakajima, H. Ishida, K. Kojima, R. Aoki, T. Oki, and S. Kobayashi, "Vision-based road-following using results of semantic segmentation for autonomous navigation," in *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 174–179, 2019.

[43] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[44] C. Nicholson, "Comparison of ai frameworks." `https://pathmind.com/wiki/comparison-frameworks-dl4j-tensorflow-pytorch`. Accessed: 2019-09-07.

[45] S. Nolen, "Gans for data augmentation." `https://medium.com/abacus-ai/gans-for-data-augmentation-21a69de6c60b`. Accessed: 2021-01-25.

[46] U. D. of Transportation Federal Aviation Administration Flight Standards Service, "Airspace," in *Pilot's Handbook of Aeronautical Knowledge*, ch. 15, pp. 15–1, Oklahoma

City, OK: the United States Department of Transportation, Federal Aviation Administration, Airman Testing Standards Branch, 2008.

[47] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Gaugan: Semantic image synthesis with spatially adaptive normalization," in *ACM SIGGRAPH 2019 Real-Time Live!*, SIGGRAPH '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[48] T. Park, M. Liu, T. Wang, and J. Zhu, "Semantic image synthesis with spatially-adaptive normalization," *CoRR*, vol. abs/1903.07291, 2019.

[49] Parrot, *Olympe Documentation*. Parrot AR.Drone.

[50] pawangfg, "R-cnn vs fast r-cnn vs faster r-cnn." `https://www.geeksforgeeks.org/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-ml/`. Accessed: 2021-02-04.

[51] X. Qi, Q. Chen, J. Jia, and V. Koltun, "Semi-parametric image synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[52] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[53] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[54] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[55] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[56] A. Sonawane, "Yolov3: A huge improvement." `https://medium.com/@anand_sonawane/yolo3-a-huge-improvement-2bc4e6fc44c5`, Mar 2018. Accessed: 2019-10-04.

[57] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 567–576, 2015.

[58] D. Support, "Revolutionize industries with your game-changing app." `https://developer.dji.com/windows-sdk/`. Accessed: 2019-09-20.

[59] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2759–2764, October 2016.

[60] N. Tijtgat, W. Van Ranst, T. Goedeme, B. Volckaert, and F. De Turck, "Embedded real-time object detection for a uav warning system," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 2110–2118, 2017.

[61] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," 2018.

[62] Wikipedia, "Data augmentation." `https://en.wikipedia.org/wiki/Data_augmentation`. Accessed: 2021-01-25.

[63] Wikipedia, "Dji." `https://en.wikipedia.org/wiki/DJI`. Accessed: 2019-09-06.

[64] Wikipedia, "Keras." `https://en.wikipedia.org/wiki/Keras`. Accessed: 2019-09-10.

[65] Wikipedia, "Long short-term memory." `https://en.wikipedia.org/wiki/Long_short-term_memory`. Accessed: 2021-03-01.

[66] Wikipedia, "Tensorflow." `https://en.wikipedia.org/wiki/TensorFlow`. Accessed: 2019-09-07.

[67] Z. Wu, C. Shen, and A. van den Hengel, "Bridging category-level and instance-level semantic image segmentation," *CoRR*, vol. abs/1605.06885, 2016.

[68] K. Yang, K. Qinami, L. Fei-Fei, J. Deng, and O. Russakovsky, "Towards fairer datasets: Filtering and balancing the distribution of the people subtree in the imagenet hierarchy," in *Conference on Fairness, Accountability, and Transparency*, 2020.

[69] X. Yang, J. Chen, Y. Dang, H. Luo, Y. Tang, C. Liao, P. Chen, and K.-T. Cheng, "Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[70] X. Yuan, L. Li, and Y. Wang, "Nonlinear dynamic soft sensor modeling with supervised long short-term memory network," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3168–3176, 2020.

[71] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," *CoRR*, vol. abs/1704.08545, 2017.

[72] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *CoRR*, vol. abs/1807.05511, 2018.

[73] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[74] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *CoRR*, vol. abs/1703.10593, 2017.