REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

GRADUATE SCHOOL OF ENGINEERING
DEPARTMENT OF AEROSPACE ENGINEERING

VISION-BASED LANDING SITE DETECTION FOR A UAV: FROM THEORY TO APPLICATION

HEDAYAH OTHMAN ISMAIL OZDEMIR

MASTER OF SCIENCE

ADANA-2023

REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF AEROSPACE ENGINEERING

VISION-BASED LANDING SITE DETECTION FOR A UAV: FROM THEORY TO APPLICATION

HEDAYAH OTHMAN ISMAIL OZDEMIR
MASTER OF SCIENCE

SUPERVISOR
Asst. Prof. Dr. Davood Asadihendoustani

ADANA 2023

VISION-BASED LANDING SITE DETECTION FOR A UAV: FROM THEORY TO APPLICATION

submitted by **HEDAYAH OTHMAN ISMAIL OZDEMIR** in partial fulfilment of the requirements for the degree of **Master of Science in the Department of Aerospace Engineering, Adana Alparslan Türkeş Science and Technology University** by,

Prof. Dr. Ahmet BEYCİOĞLU
Graduate School Director

Assoc. Prof. Dr. Tahir DURHASAN
Department Chair

Asst. Prof. Dr. Davood ASADIHENDOUSTANI
Supervisor

Thesis Committee

Asst. Prof. Dr. Davood Asadihendoustani
Adana Alparslan Türkeş Science and Technology University

Prof. Dr. Hakan Yavuz,
Çukurova University

Asst. Prof. Dr. Hürrem AKBIYIK
Adana Alparslan Türkeş Science and Technology University

Thesis Defense Date

1/12/2023

I hereby declare that all information in this thesis has been obtained and presented following in academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all information that is not original to this work.

[Signature]

Hedayah Othman Ismail OZDEMIR

ABSTRACT

VISION-BASED LANDING SITE DETECTION FOR A UAV: FROM THEORY TO APPLICATION

Hedayah Othman Ismail OZDEMIR
Master of Science, Department of Aerospace Engineering
Supervisor: Asst. Prof. Dr. Davood Asadihendoustani
Dec 2023, 86 pages

This study addresses the critical need to enhance the safety of multirotor UAVs during emergency flights by addressing the challenges of identifying suitable landing spots and avoiding collisions with obstacles. Drones encounter difficulties due to their elevated height, leading to potential collisions, inaccurate distance estimation, and weak radio signals. Researchers are diligently developing a cost-effective and efficient landing system to mitigate these issues and enhance overall safety. Key gaps in vision-based landing site detection for UAVs include adapting to varying environmental conditions, achieving real-time processing, precise obstacle recognition, ensuring robustness, and scalability, handling data anomalies, integrating with other sensors, obtaining diverse training data, complying with regulations, and maintaining cost-effectiveness. To address these challenges, the proposed model combines a U-Net, a deep Convolutional Neural Network (CNN) architecture, with a ResNet 34 backbone. The model builds a labelled aerial photo database using image segmentation techniques for CNN training, determines landing areas through 2D dataset image analysis, and plans paths from the UAV's position to the designated area centre using the A* algorithm. The entire research is implemented in Python, harnessing TensorFlow's capabilities. Impressively, the model achieves a training accuracy of 96.6% and a validation accuracy of 96.34% while utilizing approximately 80% of the dataset, comprising 1480 images, during the training phase, with a recorded training loss of 0.1034. These outcomes underscore the model's exceptional accuracy, establishing it as a cornerstone for subsequent landing area selection and path planning by the algorithm.

Keywords: Safe Landing, CNN, U-Net, Image Classification and Segmentation, ResNet.

ÖZET

İHA IÇIN GÖRÜŞ TABANLI INIŞ YERI: TEORİDEN UYGULAMAYA

Hedayah Othman Ismail OZDEMIR
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Anabilim Dalı
Danışman: Asst. Prof. Dr. Davood Asadihendoustani
Aralık 2023, 86 sayfa

Multirotor insansız hava araçları (İHA), arızalı motorlardan veya hasarlı pervanelerden kaynaklanabilen motor arızalarına oldukça yatkındır. Motor arızaları, çok motorlu İHA'nın dinamiklerini ciddi şekilde etkiler ve bu nedenle uçuş güvenliğini ve güvenilirliğini tehlikeye atar. Bu tür acil uçuş koşullarında, güvenli bir iniş İHA için hayati bir sorundur. Bu tezin amacı, görüntü işleme teknikleri ve derin öğrenme algoritmaları kullanarak uygun bir iniş alanı bulmak ve İHA'yı istenen iniş alanına doğru yönlendirmektir. Havadan çekilen görüntülerin analizi, kamera açıları, güneş ışığı, engellerin varlığı ve görüntü özelliklerinin tanınmasının zor olmasından dolayı zor bir analiz sürecidir. Arıza senaryoları gibi acil uçuş koşullarında, binalar ve ağaçlar gibi farklı engellerin bulunduğu, insan ve hayvanların bulunduğu bir alanda güvenli bir iniş alanının tespit edilmesi gerekmektedir. Bu nedenle, bu zorlukların üstesinden gelmek amacıyla, modelimizi eğitmek için Evrişimsel Sinir Ağı (CNN) mimarilerinden biri olan U-net yapısını içeren bir model kullanılmıştır. U-net mimarisi, en derin mekanizmalara ve en iyi algoritmalara sahip ResNet tabanını kullanır. Yukarıda bahsedilen hedeflere ulaşmak için, ATÜ üniversitesi kampüsünde çekilen ve hava fotoğraflarını içeren veri tabanı, CNN mimarisi kullanılmadan önce veri kümesi görüntüleri etiketlenerek segmente edilmiştir. Monoküler bir kamera kullanılarak istenen iniş alanı tespit edildikten sonra, 2D veri kümesi görüntülerinin analizine bağlı olarak İHA konumundan belirlenen alanın merkezine yol planlaması için *A** algoritması kullanılmıştır. Tüm kodlar, TensorFlow adı verilen çok kullanışlı bir kütüphanesi olan Python ortamında yazılmıştır.

Anahtar Kelimeler: Güvenli İniş, CNN, U-Net, Görüntü Sınıflandırma ve Segmentasyon, ResNet.

The evening of February 6th marked an extraordinary moment, forever altering the lives of countless individuals and stopping the passage of time for many. It is with profound gratitude that I dedicate this dissertation to all the victims and their families, as well as those who bore witness to these solemn events firsthand.

A special dedication is extended to my daughter, Nisan, who, even in the safety of my womb, cradled and secured, sensed the seismic waves trembling her world. On the thirteenth of April in the twenty-third year of the new millennium, a profound sense of salvation enveloped us, providing illumination and renewal for both me and her father. It is with heartfelt appreciation that I express my gratitude to the divine, for bestowing upon us immeasurable blessings during this journey.

ACKNOWLEDGEMENTS

This dissertation was not completed hastily; it required dedicated time, immense effort, and deep contemplation. It is with great reverence and sincerity that I present this thesis. As the saying goes, "Whoever does not thank people does not thank God."

First and foremost, I express my heartfelt gratitude to my thesis supervisor, Asst. Prof. Dr. Davood Asadihendoustani, for his unwavering guidance, and abundant suggestions throughout this endeavor. His understanding of the challenges I faced, and tireless support are truly praiseworthy. My appreciation for his contributions knows no bounds. Thank you sincerely for everything.

To my dear parents, your prayers have been a source of immense strength. Your unwavering support propelled me forward, and I am eternally grateful for that. I pray that God grants you good health and safety. My heartfelt thanks to each member of our family. Your unwavering support and love have transcended boundaries and provided me with mental well-being.

To my loving and supportive husband, who has been my steadfast companion throughout every step of this path, your unconditional love and guidance have been instrumental in my success. I am grateful for your presence in my life, and I pray that God blesses us with a lifetime of happiness, knowledge, and mutual growth.

Last but certainly not least, I would like to convey my profound gratitude to Adana University of Science and Technology for granting me this remarkable opportunity. The support provided by the university and the esteemed academic staff at the College of Aerospace and Aeronautical Engineering has been invaluable in shaping my academic journey.

I am humbled by the support I have received throughout this academic pursuit. It is with heartfelt appreciation that I express my gratitude to all those who have contributed to my success.

This thesis was financially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under the 3501 programs with project number 120M793.

TABLE OF CONTENTS

TABLE OF CONTENTS ..... **viii**
LIST OF TABLES ..... **xi**
NOMENCLATURE ..... **xii**
1. INTRODUCTION **1**
1.1. The Objective of the Dissertation 5
1.2. Importance of study 5
1.3. Critical Review 6
1.4. Methodology 7
1.5. The Outline of the Dissertation 14
2. LITERATURE REVIEW **15**
2.1. Convolutional Neural Network (CNN) 15
2.2. U-Net and ResNet34 backbone architecture 16
2.2.1. ResNet 34-backbone 18
2.2.2. U-Net architecture 20
2.2.2.1. Image Classification 22
3. Image Processing **24**
3.1. Dataset labeling 24
3.2. Image segmentation 25
3.2.1. Feature Extraction 26
3.2.2. Training data 27
3.2.2.1. Tensorflow Format and Softmax Activation Function 28
3.2.2.2. Multiply The Data Set by Smaller Patches 30
3.2.2.3. Loss function 31
3.2.2.4. Optimization Algorithm 33
3.2.2.5. Adam Optimization 34
4. Landing Site Selection **35**
4.1. Safest landing side 35
4.1.1. Largest Clearance from The Obstacle (Surrounding) 36
4.1.2. Size 37

4.1.3. Shape 38
4.1.4. Slope 38
4.1.5. Surface 39
4.1.6. Civilization 40
4.1.7. Proper Landing Area 40
5. Path Planning 42
5.1. Introduction 42
5.2. The Algorithm 43
5.2.1. Algorithm parameter 45
5.3. Path Planning 48
6. Applying Different Kinds of CNN Architectures 51
6.1. Mask-RCNN 51
6.1.1. Result of MASK-RCNN 52
6.2. U-Net architecture 59
6.2.1. Data Augmentation 60
7. RESULTS AND DISCUSSIONS 64
7.1. Prediction 64
7.2. Training result 65
7.3. Test Result 67
7.3.1. Result Of the Best Landing Site 68
7.4. Result Of the Efficient Trajectory 69
8. CONCLUSIONS 71
9. RECOMMENDATIONS 72
REFERENCES 73
CURRICULUM VITAE ..... Error! Bookmark not defined.
APPENDICES 76
Appendix A: ResNet Backbone 34 Python 76
Appendix B: Determination of Landing site 77
Appendix C: Path Planning 79

LIST OF FIGURES

Figure 1. 1. Unmanned Aerial Vehicle (UAV) With A Monocular Camera on Board 1
Figure 1. 2 Example of UAV working on Precision farming and agriculture 2
Figure 1. 3 Several examples showcasing the applications of UAVs 3
Figure 1. 4. Landing area with trajectory 4
Figure 2. 1. Structure of U-Net, ResNet 34 Backbone 17
Figure 2. 2. ResNet34 Backbone Structure 19
Figure 2. 3. U-Net Structure 21
Figure 2. 4. Examples from the dataset images labeled by CVAT 23
Figure 3.1. Examples of images captured that are used in the Segmentation 25
Figure 3. 2. Example of Some Training Images and Their Labels 30
Figure 4. 1. Options of Landing Sites 35
Figure 4.2 Safest Landing Parameters 36
Figure 4. 3. Shows The Collection of All Voronoi Polygons for Every Point in The Set. 37
Figure 4.4. Centre Point and the Box Surrounding the Landing Spot 41
Figure 5. 1. Flowchart Of Path Planning 50
Figure 6 1. Example Of MASK-RCNN Expectation Success Percentage 53
Figure 6 2. Object detection variance 54
Figure 6 3. Segmentation Variance Results 55
Figure 6.4. Total Loss 55
Figure 6.5. Indicate How Many Objects Which Have in This Training Case 56
Figure 6.6. Loss Function Result 57
Figure 6.7. Training Set Result in Classification and Segmentation 57
Figure 6.8. Indicate How Many Objects Which Have in This Training Case 58
Figure 6.9. Training Set Result in Classification and Segmentation 58
Figure 6.10. Indicate How Many Objects Which Have in This Training Case 59
Figure 6.11. Training Set Result in Classification and Segmentation 59
Figure 6.12. Shown the Data Set Has Increased To 3700 Images 61
Figure 6.13. Real Images and Its Mask with The Prediction Value 61
Figure 6.14. The Unet Accuracy Chart After Applied Data Augmentation 62
Figure 6.15. The Accuracy Result For U-Net Data Augmentation 62
Figure 7.1. Examples Of Some Output Image Expected 65
Figure 7.2. The Loss Function for Training Data 66
Figure 7.3. The Accuracy of Training and validation 66
Figure 7.4. Example of Prediction test data 68
Figure 7.5. Surrounding Region with their Numbers 68
Figure 7. 6. Path planning to the landing site 69

LIST OF TABLES

Table 1. 1. Unique Characteristics of Several CNN Architectures…….07

Table 1.2. Advantages and drawbacks of select CNN architectures along with typical applications……...…08

Table 6 1. Some classes and their predictions of success percentage 53
Table 6 2. The Average of Success Rate for Each Object 54

NOMENCLATURE

UAV	: Unmanned Aerial Vehicles
CNN	: Convolutional Neural Network
ResNet	: Residual Neural Network
RGB	: Red, Green, and Blue
FC	: Fully Connected
ReLU	: Rectified Linear Unit
CVAT	: Computer Vision Annotation Tool
DSC	: Dice Similarity Coefficient
Adam	: Adaptive Moment Estimation
AdaGrad	: Adaptive Gradient Algorithm
RMSProp	: Root Mean Square Propagation
GVD	: Generalized Voronoi Diagram
mAP	: Mean Average Precision
IoU	: Intersection Over Union
CCD	: Charge-Coupled Device
CMOS	: Complementary Metal Oxide Semiconductor
DDA	: Digital Differential Analyzer

1. INTRODUCTION

The utilization of unmanned aerial vehicles (UAV) has experienced an unpredictable surge in recent years. Multi-rotor drones have gained significant attention among these UAVs, witnessing rapid development and widespread adoption. However, this increased usage has brought to light a critical concern: ensuring flight safety and reliability in aerial applications. One of the primary safety issues faced by multi-rotor drones is their susceptibility to motor failures, which can lead to risky operations or even crashes. To address this, researchers are focusing on developing autonomous landing systems that can safely retrieve and land malfunctioning drones at their intended landing spots.

In this dissertation, we present a novel architecture for such a landing system, which uniquely relies solely on image processing from the vehicle's onboard camera. Rather than using markers and GPS signals, we propose a method to distinguish safe landing zones from hazardous ones using simple RGB images captured by a single UAV camera. To be specific, the study delves into the concept of a monocular camera, a single-lens camera system capable of capturing images or videos. The architecture aims to swiftly identify suitable emergency landing areas based on the analysis of these images, as depicted in Figure 1.1.

Figure 1. 1. Unmanned Aerial Vehicle (UAV) with a Monocular Camera on Board

Quadcopters, unmanned aerial vehicles (UAVs) fitted with a single-lens camera, have experienced a remarkable surge in adoption across diverse fields. Their adaptability, cost-efficiency, and accessibility have fuelled their increasing popularity in recent years. These UAVs have found practical use in various commercial and non-commercial domains. Here are some common applications where UAVs are frequently employed:

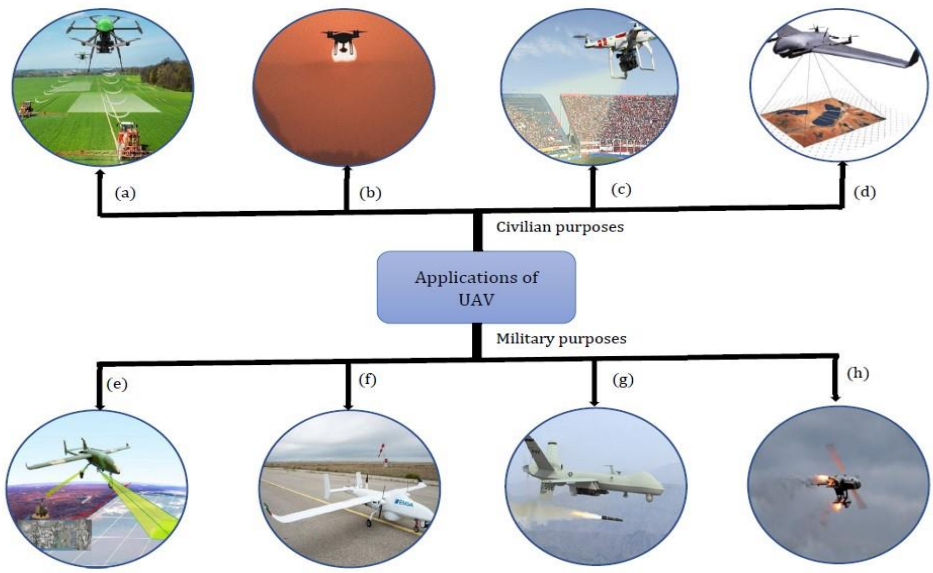
- Aerial Photography and Videography have undergone a transformative revolution with the integration of UAVs equipped with high-resolution cameras. Various sectors, including filmmaking, real estate, tourism, and event coverage, have immensely benefitted from their application (Çıkmak et al., 2023). UAVs bring a fresh dimension to the table by providing unique perspectives and angles, capturing awe-inspiring images that were once attainable only through costly or helicopter-based equipment.

Figure 1. 2. Example of UAV Working on Precision Farming and Agriculture

- Precision farming and agriculture have undergone a profound transformation, thanks to UAVs, as illustrated in Figure 1.2. These unmanned aerial vehicles have enabled modern agricultural practices by offering valuable capabilities. Equipped with multispectral or thermal cameras, UAVs can effectively monitor pest activity and regulate irrigation systems. By providing essential data, these UAVs aid farmers in making informed decisions, enhancing agricultural productivity, and reducing resource wastage.
- Environmental Monitoring: UAVs are essential to attempts to conserve the environment. They can keep an eye on criminal activity, wildlife populations, and deforestation in far-off places. UAVs with specific sensors can measure air and water pollution as well as track changes in ecosystems (Giordan et al., 2020).

- Infrastructure Inspection: Using UAVs to check infrastructure like bridges, electricity lines, pipelines, and buildings is a safe and affordable option. They can rapidly and effectively take high-resolution photos and videos that may be used to spot damage, corrosion, or structural problems. This reduces the need for human inspections and improves worker safety (Giordan et al., 2020).
- UAVs are utilized in search and rescue operations to find missing people or survivors in disaster-stricken areas. They can cover large regions and offer real-time situational awareness since they have GPS and thermal imaging cameras.
- Research & Scientific Studies: UAVs are widely employed in scientific research, including studies of the environment, the monitoring of wildlife, investigations of the atmosphere, and surveys of the earth's geology. Researchers can obtaining important information and carry out studies more successfully because of their capacity to access remote and inaccessible regions.
- Recreational and Hobbyist Use: Aerial photography, racing, and exploration are just a few of the recreational uses for UAVs that have gained popularity among hobbyists and enthusiasts. This broad use has sped up drone technology advancements and encouraged user innovation.

As depicted in Figure 1.3 (Pham et al., 2022), drones can perform and apply numerous applications.



Sources: (Pham et al., 2022)

Figure 1. 3. Several Examples Showcasing the Applications of UAV's

To achieve the different missions mission described above, UAVs should include the desired hardware and software. The hardware consists of the boards, sensor, and different electronic parts. In the following, some important parts of the multirotor UAVs are listed:

- Flight Control: To perform desired flight maneuvers manoeuvres (Huang et al., 2015), the quadcopter is controlled by a flight controller that accepts input from the operator or an autonomous system and modifies the motor speeds accordingly.
- Monocular Camera: The monocular camera records images of the surrounding area. It frequently uses a CMOS or CCD sensor (Dhall, 2018) to capture images or videos in real time real-time. Resolution, frame rate, and lens focal length are examples of ordinary qualities.
- Image processing: The monocular camera's visual data can be analyzed analysed onboard the UAV or sent to a ground station for additional examination. Algorithms for image processing can improve image quality, extract features, find objects, or carry out other activities requiring computer vision (Bechini et al., 2023).

The focus of this thesis revolves around the utilization of a monocular camera for the vital tasks of landing site detection and path planning to the designated landing area.

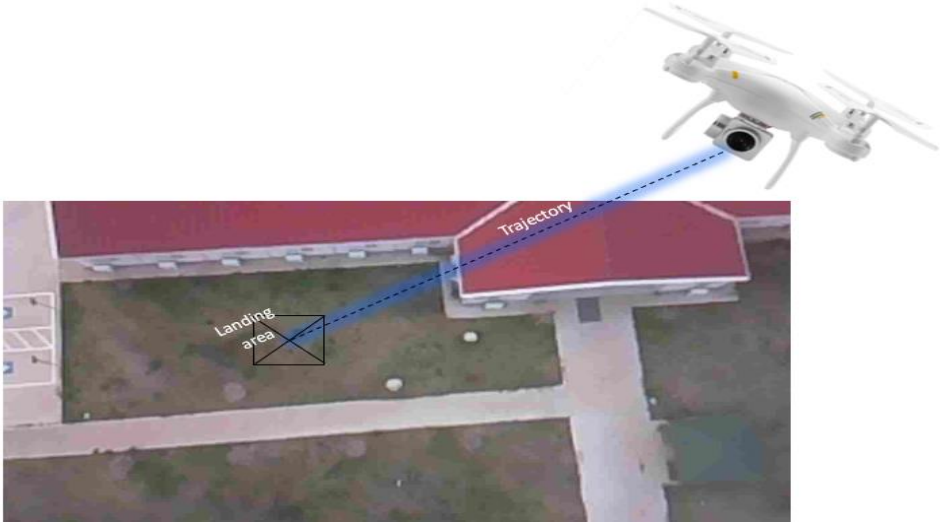


Figure 1. 4. Landing Area with Trajectory

To achieve this goal, a Convolutional Neural Network (CNN) is employed as an efficient tool for classifying and segmenting image data to discern various elements, building upon successful

experiments conducted by numerous researchers (Preethi et al., 2021). Subsequently, the images classified in this manner are used to map out a route for the drone, as depicted in Fig.1.4.

In image classification tasks, Convolutional Neural Networks (CNNs) play a prominent role. CNNs are designed to automatically learn hierarchical image representations by utilizing interconnected layers of nodes known as neurons. Each neuron applies a sequence of operations, including convolution, non-linear activation functions, pooling, and fully connected layers, to make accurate predictions. Once the model is trained, it can classify unseen images by inputting them into the network, which will produce either the predicted class or a probability distribution across different classes. Model performance is assessed using metrics such as accuracy, precision, recall, and F1 score (Goutte & Gaussier, 2005).

To determine the appropriate landing point, we compute the midpoint coordinates of each object's bounding box. This method enables us to establish the center point, considering the bounding box's top-left corner coordinates (x, y) along with its width and height values. Finally, for pathfinding within an image, we have implemented the A* search algorithm. This algorithm efficiently navigates the image, facilitating the identification of an optimal path from a starting node to a target node.

1.1. The Objective of the Dissertation

The primary objective of this project is to develop an image-based landing system for drones, leveraging deep learning techniques. The core focus is on using the drone's onboard camera to detect objects and identify safe landing sites. This involves exploring semantic image segmentation frameworks to pinpoint secure landing zones. The project also emphasizes the development of lightweight models to reduce the dependency on multiple sensors.

The driving force behind this research is the imperative to enhance drone safety during landings, addressing potential emergencies. Our ultimate goal is to minimize both human casualties and financial losses. To achieve this, we adopt a dataset-driven approach using RGB images, specifically avoiding the resource-intensive nature of video data. Through this approach, we aim to improve landing safety while optimizing energy consumption and resource utilization.

1.2. Importance of study

Drones have gained widespread adoption in numerous countries, finding applications in both commercial and military sectors, which has contributed to their increasing popularity. Their use

is no longer confined to remote or secluded areas; they are now integrated into various daily tasks, including commercial transportation, photography, and more. The importance of this study is to enhance the safety of multirotor vehicles (UAVs), in emergency flight situations. When a drone encounters an emergency situation and needs to land in an area it becomes essential to identify a suitable landing spot. Moreover, ensuring a landing without collisions with obstacles plays a role in achieving a successful emergency landing.

One apparent issue arises from the drone's elevated height, making it difficult to control and leading to collisions with objects. These collisions convert potential energy into kinetic collision energy, among other problems like inaccurate distance estimation and weak radio signals. Consequently, scientists have been working diligently to develop a safer landing system that requires fewer components, reducing costs and extending the system's operational range.

Among the notable applications of deep learning, the Convolutional Neural Network (CNN), also known as ConvNet, has received significant attention (Indolia et al., 2018).

1.3. Critical Review

Vision-based landing site detection for Unmanned Aerial Vehicles (UAVs) is integral to achieving autonomous flight capabilities and has gained substantial attention in recent years. Convolutional Neural Networks (CNNs) have emerged as potent tools for image analysis, finding extensive application in this domain. This comprehensive review aims to offer an overarching perspective on the current state-of-the-art in this field, accentuating pivotal advancements, persistent challenges, and promising prospects. CNNs have proven their mettle in object detection and image classification tasks, rendering them apt for landing site recognition.

Researchers have capitalized on established CNN architectures like VGG, ResNet, and MobileNet, fine-tuning them for specialized landing site detection (Khan et al., 2020). Transfer learning has been a prevalent strategy, enabling models to glean insights from broad image datasets and adapt them to the nuances of landing site identification. However, the availability of substantial and diverse datasets for CNN training remains paramount, although creating annotated datasets for this purpose presents formidable challenges (Al-qurishi et al., 2021). Synthetic datasets have been utilized to augment the paucity of real-world data, although concerns regarding model generalization persist. Real-time processing is imperative for UAV applications, prompting researchers to concentrate on optimizing CNN models for

computational efficiency through techniques like model quantization and pruning. Robustness to environmental variability, encompassing changes in lighting, weather, and terrain, is a critical concern. Some studies have integrated supplementary sensors such as LiDAR and infrared to augment visual data and enhance performance in challenging conditions (Ameli et al., 2022). Ensuring the generalization of CNN-based landing site detection models across diverse UAV platforms and configurations is essential. To this end, domain adaptation and transfer learning techniques have been explored to enhance model versatility (Telli et al., 2023).

Furthermore, the human-UAV interaction aspect is gaining prominence, with research focusing on how humans and AI systems collaborate during the landing process, including human intervention (Mualla, 2021). As UAVs progressively transition towards greater autonomy, ethical and safety considerations loom large. Ensuring the reliability and safety of operations stands as an overarching priority. In conclusion, while vision-based landing site detection employing CNNs has witnessed substantial advancements, significant challenges persist in terms of dataset availability, real-time processing, environmental robustness, and model generalization. Future research efforts should be directed towards overcoming these hurdles, enhancing human-AI collaboration, and upkeeping ethical and safe UAV operations (Fakhraian et al., 2023). Remaining abreast of the latest developments in CNN architectures and training methodologies is indispensable for the continued progression of this field.

1.4. Methodology

YOLO (You Only Look Once), CNN (Convolutional Neural Network), and RNN (Recurrent Neural Network) represent prominent deep learning architectures extensively employed in the realm of image processing and object recognition (Redmon et al., 2016). It's important to clarify that these architectures are indeed parametric models. During their training phases, they dynamically adjust and fine-tune a set of parameters, which includes weights and biases, through techniques like backpropagation and gradient descent. These learned parameters are then leveraged for extracting features from input data, such as images, and making predictions. In essence, their parametric nature empowers them to adapt and generalize to a wide range of object recognition tasks and image processing challenges, making them foundational components in the field of computer vision and machine learning.

Convolutional Neural Networks (CNNs) have completely changed the way that images are processed, allowing for considerable improvements in a wide range of tasks like image segmentation, detecting objects, and classification. An overview of significant studies and

advancements in the field of image processing with CNNs is given in this survey of the literature. It covers a wide range of subjects, such as CNN applications in image processing, network structures, training methods, and optimization strategies. In the field of image classification tasks Table 1.1 provides an overview of CNN architectures that are commonly used.

Table 1. 3. Unique Characteristics of Several CNN Architectures

CNN Structure	Architecture	Distinctive Attributes
AlexNet	Conv - Conv - Pool - Conv - Conv - Pool - FC - FC	Introduces deep CNNs, ReLU activation, Dropout
ResNet	Residual blocks with skip connections	Solves vanishing gradient, very deep networks
UNet	Contracting and Expanding paths	Semantic segmentation, Skip connections
Mask-RCNN	UNet architecture with ResNet 34 as backbone	Combines UNet and ResNet for segmentation
U-Net and ResNet34 backbone	Faster R-CNN with Mask branch	Object detection and instance segmentation

The paper by Krizhevsky and Hinton introduced the use of deep convolutional neural networks for ImageNet classification, leading to a significant breakthrough in the field (Krizhevsky & Hinton, n.d.). Their proposed architecture, AlexNet, made remarkable progress in the ImageNet Large-Scale Visual Recognition Challenge, igniting a resurgence of convolutional neural networks in computer vision (Arge & Mage, 2015). To address the degradation problem in very deep networks, ResNet was introduced by Ozturk et al. (2020). ResNet utilized residual connections, surpassing other competitors in the ImageNet challenge, and quickly becoming popular for various image processing tasks.

ResNet's potential to address the vanishing gradient problem is noteworthy, given that this issue arises when gradients diminish significantly during backpropagation within deep neural networks, hindering weight updates in early layers and impeding training progress (Shah et al., 2016). ResNet tackles this challenge by introducing skip connections or residual blocks, facilitating smoother gradient flow throughout the network (Rayhan & Kinzler, 2023). This

approach enables the network to learn residual functions, enhancing weight optimization. Nevertheless, it is important to note that ResNet does not completely eliminate the vanishing gradient problem but rather mitigates it to a considerable extent. In summary, ResNet plays a significant role in alleviating the vanishing gradient problem, thereby enhancing the efficacy of training very deep neural networks compared to some earlier architectures (Turgut, 2020).

For biological image segmentation tasks, the UNet architecture was specifically designed. It incorporated skip connections to carry contextual information from earlier layers to subsequent layers, enabling precise object localization (Redmon et al., 2016). The Mask R-CNN extended the Faster R-CNN object detection framework by adding instance-level segmentation. In addition to predicting bounding boxes, it introduced a branch for generating pixel-level masks, thereby achieving accurate instance segmentation (Zhang et al., 2016).

Overall, these advancements in deep learning architectures have revolutionized the fields of image classification, object detection, and biological image segmentation, enabling significant progress in these areas.

Previous studies provide a glimpse into the advancements in CNN-based image processing. However, it is important to note that these studies are not exhaustive due to the rapid development of the field. New methods and architectural designs are continuously emerging, contributing to the evolving landscape of CNN-based image processing.

In summarizing the differences between CNN architectures discussed in Table 1.2 compared their strengths and weaknesses. After evaluation we concluded that utilizing Unet with a ResNet 34 backbone showed promising results, in this thesis.

Table 1. 4. Advantages and Drawbacks of Select CNN Architectures Along with Typical Applications.

CNN Structure	*Pros.*	*Limitations*	*Domains*
AlexNet	**Pioneering Architecture**: AlexNet pioneered deep learning architectures,	**Simpler Architectures Surpassed**: Newer architectures like ResNet outperform	AlexNet was innovative initially but has been overtaken by newer architectures such as

	promoting deep CNN usage. **Revolutionized ImageNet**: Its ImageNet success in 2012 transformed image recognition accuracy. **Introduced Convolutional Layers**: It proposed stacking convolutional layers, a modern CNN foundation. **Parallelization**: Used GPUs for training, showcasing parallel computing potential. **Influence on Future Architectures**: AlexNet influenced CNN evolution, enabling deeper networks and improved activations.	AlexNet in accuracy and efficiency. **Limited Depth & Overfitting**: AlexNet's shallowness hampers complex feature capture, leading to overfitting, particularly on intricate datasets. **Outdated Features**: Some introduced features (e.g., local response normalization) are no longer prevalent in modern architecture. **Less Generalization**: AlexNet's generic design may hinder effective task-specific generalization.	ResNet. Its primary role now is for education to grasp deep learning fundamentals or as a foundation for experimenting with image classification on simpler datasets.
ResNet	**Deep Networks**: ResNet introduced skip connections	**Complexity**: Deeper architectures can be computationally	ResNet excels in diverse computer vision tasks,

	that mitigate the vanishing gradient problem, allowing for the creation of much deeper networks. **Better Training**: Easier training of very deep networks due to skip connections. **State-of-the-art Performance**: ResNet variants have consistently achieved top performance in various computer vision tasks. **Transfer Learning**: Pre-trained ResNet models are readily available and can be fine-tuned for specific tasks.	expensive and memory intensive. **Overfitting**: If not carefully managed, very deep networks can still suffer from overfitting.	particularly in scenarios needing depth. It performs well in image classification, object detection, and tasks demanding accuracy and deep architectures. Transfer learning using pre-trained ResNet models is also potent
UNet	**Semantic Segmentation**: UNet is specifically designed for semantic segmentation tasks.	**Limited to Segmentation**: UNet is specialized for segmentation tasks and might not be optimal for other	UNet targets semantic segmentation tasks, like medical imaging or scenarios demanding pixel-level labeling. It's

	Skip Connections: The architecture uses skip connections that aid in combining contextual information from different scales. **Detailed Outputs**: UNet produces detailed segmentations, useful in medical imaging and other tasks requiring fine-grained results	computer vision tasks.	valuable for object segmentation, preserving fine details in segmented regions
Mask-RCNN	**Instance Segmentation**: Offers both object detection and instance segmentation in a single architecture. **Precise Localization**: Can accurately locate object instances along with their segmentations. **Multiple Tasks**: Supports object detection, segmentation, and	**Complexity**: Mask R-CNN is more complex due to handling multiple tasks, potentially requiring more training time and computational resources. **Resource Intensive**: Can be computationally demanding, limiting real-time applications on resource-constrained devices.	Mask R-CNN suits intricate tasks like object detection and instance segmentation. It's ideal for precise object boundaries and detailed segmentation masks. Commonly used in image and video object detection for individual instance identification and segmentation

	even key point estimation tasks.		
U-Net and ResNet34 backbone	**Leveraging Both Architectures**: Combines the segmentation prowess of UNet with the feature extraction capabilities of ResNet. **Improved Feature Extraction**: ResNet's deeper architecture can enhance the feature extraction process in the UNet.	**Complexity**: Combining two architectures can make the resulting model more complex. **Computational Demands**: The combined model might require more computational resources. **Risk of Overfitting**: The heightened model complexity can increase the risk of overfitting, particularly with limited data, necessitating regularization and data augmentation for mitigation. **Challenging Skip Connections**: Integrating skip connections between ResNet-34 and UNet can be complex, requiring effective optimization for seamless information flow and spatial coherence.	This architecture merges UNet's segmentation prowess with ResNet's robust feature extraction. Suitable for tasks needing detailed segmentation and precise feature extraction, like medical image analysis detecting subtle features.

1.5. The Outline of the Dissertation

This dissertation was structured as follows:

- Chapter 2: Literature Review

  This chapter provides a comprehensive review of the Convolutional Neural Network (CNN), U-Net architecture, and ResNet34 backbone.

- Chapter 3: Methodology

  In this chapter, we discuss the methods employed for label images, patchify, image classification, image segmentation, as well as the preparation of Training and Testing data.

- Chapter 4: Finding the Suitable Landing Spot

  Building upon the methods outlined in Chapter 3, this chapter focuses on identifying the appropriate landing site and devising an efficient trajectory to reach the landing spot, including relevant details.

- Chapter 5: Efficient Trajectory

  This chapter delves deeper into the development and analysis of an efficient trajectory for the drone.

- Chapter 6: Applying Different Kinds of CNN Architectures

  This chapter explores other model architectures used in related works and presents several case studies.

- Chapter 7: Final Result

  Here, we present the conclusive outcomes and findings of this dissertation.

- Chapter 8: Conclusion

  In this chapter, we summarize the key findings and draw conclusions based on the research conducted.

- Chapter 9: Recommendations

  The final chapter provides valuable recommendations based on the dissertation's results and insights for further exploration and improvement in the future.

2. LITERATURE REVIEW

In this chapter, we will delve into the U-Net and ResNet34 deep learning architectures, which played a pivotal role in this thesis and are thoroughly explored in the literature review. ResNet34, a variant of the renowned ResNet architecture, stands out for its deep layer structure, while U-Net, a convolutional neural network (CNN) architecture, is frequently employed for semantic segmentation tasks.

Throughout the discussion, we will cover the fundamental characteristics, advantages, and limitations of both architectures, along with providing examples of their applications in various fields. Deep learning has revolutionized computer vision problems, and U-Net and ResNet34 have emerged as widely used architectures in this domain. This section will briefly elaborate on the significance of semantic segmentation and the rationale behind choosing U-Net and ResNet34 for the research at hand.

2.1. Convolutional Neural Network (CNN)

In recent years, Convolutional Neural Networks (CNN) have demonstrated remarkable achievements in image processing and sound recognition (Albawi et al., 2017). These networks are distinguished by their deep information structure and an impressive capacity for comprehensiveness, functioning in a manner akin to the human brain's neural network (Indolia et al., 2018).

One of CNN's key strengths lies in its ability to retain positional information while minimizing the number of trainable parameters, making it particularly well-suited for image classification tasks. It is worth noting that CNN is a specific subclass of neural networks (Pollestad, 2019). A widely applied application of CNN is the identification of blurry and unknown images (Saravanan et al., 2019). The significance of this network lies in its capability to identify objects of interest regardless of their position within the image. Additionally, as CNN progresses through deeper layers, it progressively reveals more complex features. For instance, the initial layers might expose angles, followed by simpler shapes in the subsequent layers, and eventually higher-level features and structures (Albawi et al., 2017).

The core of CNN's data processing lies in its utilization of convolutional operations, which aids in extracting essential parameters from vast datasets (Sarıgül et al., 2019). Moreover, CNN can

discern patterns and structures without relying on any pre-established image processing techniques or having prior knowledge about the cause of the visual effects (Xu et al., 2014).

In summary, CNN excels at constructing feature formats for images by preserving image regions, extracting regular-shaped segments, and subjecting them to non-linear processing (Bosch et al., 2006). Its adaptability, accuracy, and ability to uncover relevant details in images have made it a powerful tool in various fields of image recognition and classification. In general, a convolutional neural network contains an input layer, convolution, and pooling layers as single or multiple blocks, in addition, to fully connected (FC) layers with output layers (Ghosh et al., 2019). On the other hand, CNN has different types of architectures such as U-Net and ResNet 34 backbone which are considered in this thesis.

2.2. U-Net and ResNet34 backbone architecture

The integration of ResNet-34 as a backbone in the U-Net architecture establishes a vital connection between the two. By incorporating a ResNet-34 backbone, U-Net can leverage the robust feature extraction capabilities of ResNet-34. This enables the U-Net to use ResNet-34's feature extraction process in place of its conventional contracting path, which typically involves multiple convolutional and pooling layers.

The ResNet-34 backbone brings several advantages to the U-Net. Firstly, it facilitates deep feature extraction, allowing the system to identify intricate patterns within the input images effectively. Additionally, the ResNet-34 backbone's capacity for recognizing complex patterns enhances the U-Net's segmentation performance.

In the U-Net architecture, the extensive path, comprising transposed convolutions and skip connections, can then take the output feature maps from the ResNet-34 backbone as input. This configuration allows the U-Net to benefit from the precise segmentation achieved by ResNet-34 while also gaining access to the contextual information gathered through the skip connections. For a visual representation of this structure, please refer to Figure 2.1, which illustrates the U-Net with the integrated ResNet-34 backbone. This combined architecture maximizes the strengths of both models, resulting in a powerful and effective system for various tasks, such as image segmentation and recognition.

The U-Net architecture incorporates a method that involves replacing the feature encoding process's backbone with a 34-layer ResNet structure, which significantly improves the model's training process. ResNet34 has previously been trained on a vast dataset of over one hundred thousand images across approximately two hundred categories. As the feature encoding layers increase, the depth of feature representation grows, enabling the model to perform more sophisticated analysis in both image classification and segmentation tasks.

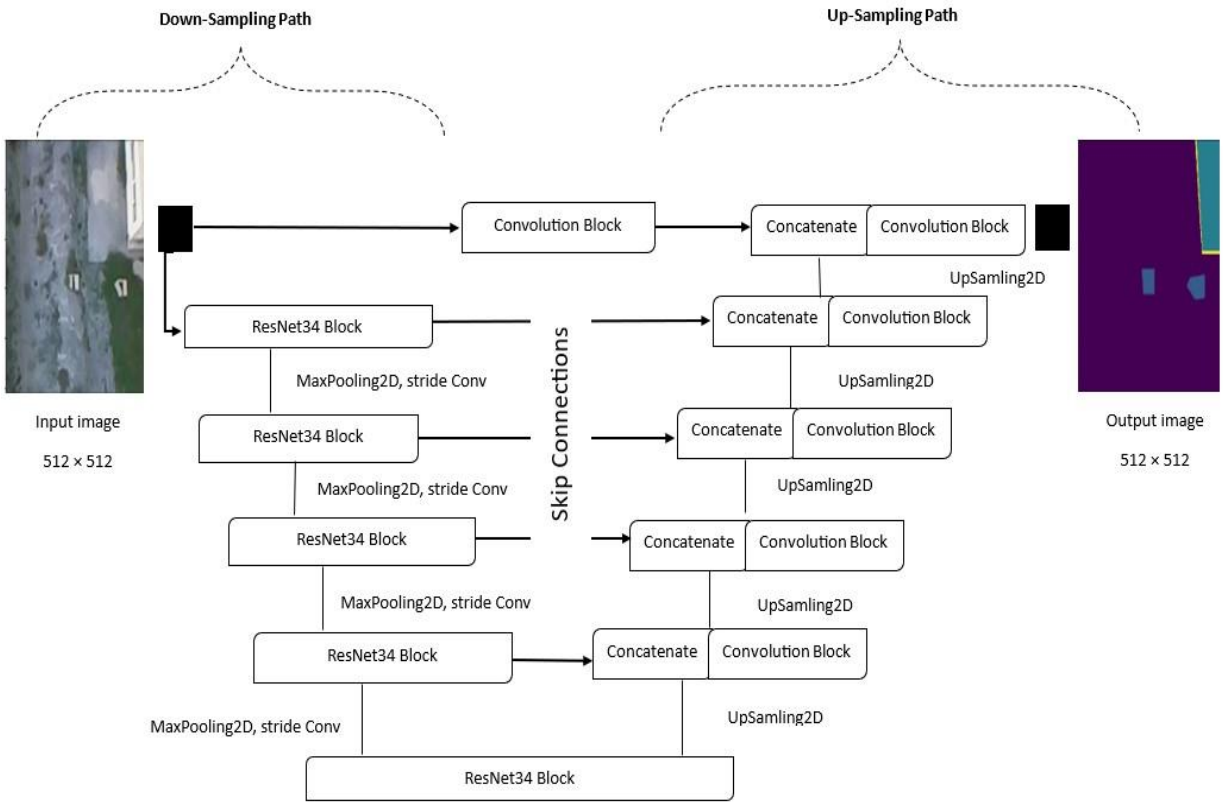


Figure 2. 1. Structure of U-Net, ResNet 34 Backbone

This structural approach aims to create rich features within the image data by studying the objects present. ResNet34 serves as the backbone of the contracting path, effectively reducing the image dimensions while simultaneously increasing its depth (Nathania et al., 2022). By employing this combination, the model becomes proficient in distinguishing itself from other segmentation networks.

Moreover, this fusion approach expedites the pre-learning process, enabling the model to extract additional features that align with our objectives (Reziapov, n.d.). Furthermore, it enhances the efficiency of U-Net classification (Li et al., 2022) and reduces the execution time (Neven & Goedemé, 2021). This integration of ResNet34 within the U-Net architecture brings

notable improvements, making the model more adept at handling complex image analysis tasks and yielding promising results for various applications.

2.2.1. ResNet 34-backbone

The ResNet architecture, introduced by He et al. in 2015, addresses the issue of vanishing gradients in deep neural networks. One specific variant, ResNet34, consists of 34 layers and offers several advantages, such as improved feature extraction, increased model depth without performance degradation, and enhanced training convergence. ResNet34 has demonstrated impressive results, particularly in image classification, object detection, and picture recognition tasks within computer vision applications. However, while ResNet34 brings significant benefits, it also has some limitations. These include higher computing costs and the potential for overfitting when training data is sparse. Despite these drawbacks, ResNet34 remains widely utilized due to its outstanding performance in various computer vision tasks.

This method leverages the encoder and decoder units, relying on weights assigned to specific categories for classification. By utilizing these weights, the decoder unit can prioritize the most relevant and weighted categories and link them to the corresponding parts of the input data (Shahabi & Ghorbanzadeh, 2022).

Recently, ResNet34 has also been explored for text prediction, further showcasing its versatility beyond image classification. For a visual representation of the ResNet34 structure, please refer to Figure 2.2. Overall, ResNet34 continues to be a popular and effective choice in deep learning and different neural network applications, offering improved training stability and performance, especially in tasks related to image and text analysis.

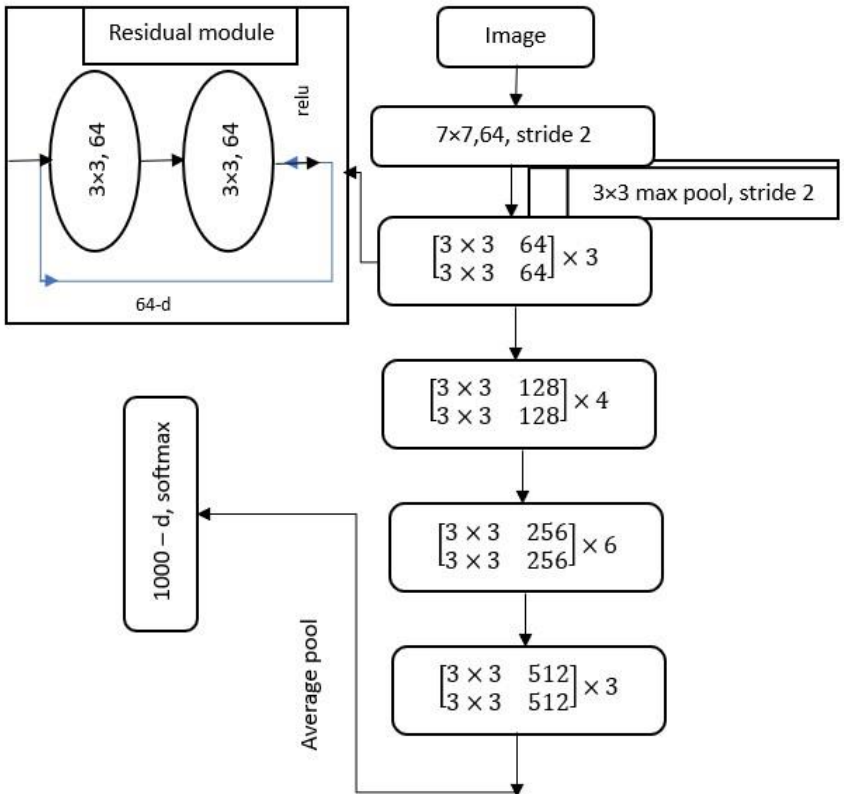


Figure 2. 2. ResNet34 Backbone Structure

ResNet34 adopts a unique approach of capturing residuals from each layer and utilizing them in subsequent interconnected layers. This architecture comprises a 34-layer convolutional neural network, pre-trained on a substantial dataset named ImageNet, encompassing more than a hundred thousand images spanning approximately two hundred classes. By leveraging this pre-training, the network effectively learns and exploits residual information across its layers, leading to enhanced overall performance and representation capabilities.

In this model, all nine classes are given equal weight, signifying that they are of equal importance. The initial convolution layer consists of 64 filters with a kernel size of 7×7, followed by a max-pooling layer. This specific stage is repeated twice to extract essential features. Moreover, conv2_x employs pairs of pooling layers followed by convolution layers, facilitating communication between the pooling layers using 3×3 kernel size, and this is repeated three times. A 128-filter layer is then utilized and repeated approximately four times, maintaining a consistent pattern. Finally, the model incorporates the SoftMax function and an average pooling layer before reaching the output layer, making it suitable for its designated tasks.

In the ResNet-34 architecture, the mention of "Matrices with 3 by 3" pertains to the use of 3x3 convolutional kernels or filters within specific layers. These kernels are employed to convolve over input data, extracting essential features as they slide across the data. This feature extraction process is fundamental to understanding complex patterns in the input. Additionally, the reference to "256 Values" signifies that these convolutional kernels possess 256 channels or filters. Each channel captures distinct aspects of the input data, resulting in each convolutional layer generating 256 feature maps as output. These feature maps encode diverse feature information derived from the input. In essence, these architectural characteristics of ResNet-34, including the utilization of 3x3 convolutional kernels with 256 channels, play a pivotal role in enabling the network to effectively learn and represent intricate visual patterns in data, making it highly proficient in tasks such as image classification and segmentation.

ArchitectureResNet34's innovative design of capturing residuals and its well-structured architecture, along with its pre-trained capabilities, contribute to its remarkable performance and effectiveness in a variety of image recognition and classification tasks.

2.2.2. U-Net architecture

The U-Net architecture, introduced by Ronneberger et al. in 2015, has garnered significant attention for its exceptional performance in semantic segmentation tasks. This subsection provides a description of the unique structure of U-Net, featuring an encoder-decoder design with skip connections. Among CNN architectures, U-Net stands out as a significant improvement for image segmentation, making it a widely adopted tool for various segmentation tasks (Siddique et al., 2021).

U-Net is a fully convolutional network that incorporates an encoder in the contracting path and a decoder in the expanding path, comprising multiple sub-layers in each path (J. Huang & Nowack, 2020). The encoder is responsible for reducing the spatial dimensions of the input, while the decoder aims to upscale the feature maps to the original input size. The integration of skip connections between these two paths allows the model to efficiently propagate detailed spatial information between the encoder and decoder, enhancing the segmentation results (Ozturk et al., 2020). U-Net is also symmetrically structured, making it well-balanced and effective in handling various segmentation tasks (as shown in Figure 2.3).

In summary, the U-Net architecture's encoder-decoder design, coupled with skip connections, has solidified its position as an influential and powerful tool for semantic segmentation, contributing to its widespread use in numerous image segmentation applications.

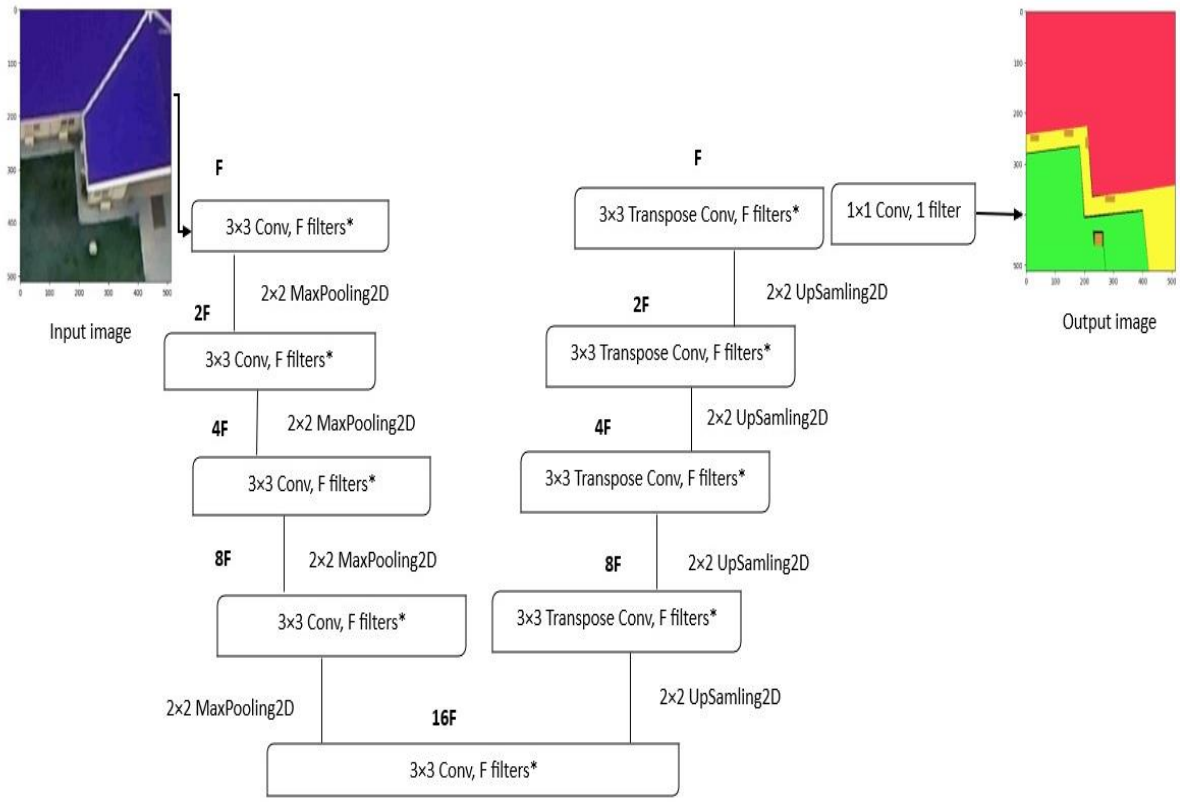


Figure 2. 3. U-Net Structure

U-Net offers several advantages, making it a popular choice for various tasks. It excels in handling limited training data, efficiently capturing small features, and providing interpretability through skip connections. However, like any model, it has some limitations, such as difficulties in handling class imbalance and precisely delineating boundaries, despite its widespread usage.

Unlike classification processes, U-Net's structure relies on the final output of the neural network, as it requires an explanation of the discrimination mechanism learned by the encoder on the pixel level. To tailor the model to our specific needs, we trained it using the ResNet-34 algorithm, followed by a basic U-Net architecture, specializing in the training process for nine classes corresponding to the objects in the dataset. To accommodate multi-class scenarios where members of the class may have more than two labels, we employed the SoftMax activation function before the output layer. Additionally, to address the vanishing gradient issue and ensure efficient computation, we used ReLU in the hidden layers.

In recent years, U-Net architecture has demonstrated remarkable success in various domains, such as satellite imagery, industrial quality control, and medical image analysis. Its effectiveness and adaptability have made it an invaluable tool in these domains and beyond, showcasing its potential in addressing diverse real-world challenges.

2.2.2.1. Image Classification

With reference to the experiments conducted in the antecedent stages of this research, the utilization of the UNet architecture was favored for the purpose of image classification. This choice stemmed from U-Net's proficiency in the classification and partitioning of images. UNet demonstrates a remarkable capability to classify individual pixels within an image, thereby assigning precise labels or categories to each pixel. This pixel-level classification precision empowers UNet to discern and delineate objects or regions of interest within an image with notable accuracy. Notably, U-Net's prowess in semantic segmentation positions it as a valuable asset across diverse applications, encompassing medical image analysis, autonomous navigation for unmanned aerial vehicles (UAVs) and self-driving automobiles, as well as image-to-image translation tasks, among others.

This thesis centers around image classification using images captured by UAVs through their onboard RGB cameras. The images have a standardized resolution of 1280x720 pixels. Figure 2.4 presents examples of these captured images.

The image classification process involves identifying the primary features present in the images. In other words, the system can recognize and categorize the content of each image into one of nine classes: background, grass, building, misc, people, road, feature, trees, and vehicles, which are specific to this project. By effectively classifying the images, this project aims to gain valuable insights from the data obtained through UAV imagery.

Figure 2. 4. Examples from The Dataset Images

The key idea behind CNNs is the use of convolutional layers. These layers consist of filters (also called kernels), which are small windows that slide across the input image to perform convolution operations. The convolution operation involves element-wise multiplication of the filter with the corresponding patch of the input image, followed by summing up the results to produce a single value in the output (feature) map.

In order to enhance the labeling phase, leveraging the hierarchical structure of the CNN classifier proves beneficial for improving feature exploration methods. This artificial neural network is capable of learning to classify images by utilizing multiple convolutional layers, as demonstrated (Al-Najjar et al., 2019).

3. Image Processing

Image processing plays a pivotal role in convolutional neural networks (CNNs), a class of deep learning models extensively employed in computer vision tasks. In this thesis, we outline several essential steps that contribute to achieving our goal.

3.1. Dataset labeling

In this section, "Labelling the Specific Objects" refers to the process of annotating or tagging individual objects or elements within a dataset, as exemplified in this thesis through the annotation of objects like cars, trees, or buildings in computer vision tasks. Consequently, the algorithm extends its knowledge base from these meticulously labeled instances. In this context, "Generalization" entails enabling a machine learning or AI system to discern analogous objects or patterns in previously unencountered data. The objective is to instruct the system in applying the labels acquired from these specific instances to novel scenarios, even when they do not precisely replicate the originally annotated examples. This dual process involves both the meticulous labeling of specific instances and empowering the system to extrapolate this knowledge for predictions or classifications within a broader context. This amalgamation is a fundamental tenet of supervised machine learning, where labeled data serves as the foundation for training models capable of generalizing their understanding to make predictions regarding fresh, unlabeled data.

In the context of this work, one of the primary and crucial steps in supervised machine learning is the image labeling process. This step is instrumental in achieving image segmentation tasks. However, it can be time-consuming due to its significance in the implementation. The accuracy of image labeling directly impacts the quality of the results and the model's performance. The process of presenting examples to the model during training allows it to learn effectively, making the quality of the labels crucial.

For this project, the Computer Vision Annotation Tool (CVAT) is utilized as an open-source web application for image annotation. CVAT facilitates dividing the project into manageable tasks, allowing efficient collaboration among team members for completing the project. The application offers various annotation options based on the image dataset's characteristics and the specific purpose of the annotation. These annotations include key-points and landmarks, bounding boxes, 3D cuboids, polygonal segmentation, semantic segmentation, lines, and splines.

Given that the objects within the dataset possess varying shapes and are not confined to rectangles or squares, a polygonal segmentation tool is employed. This approach enables more accurate identification of objects and their precise locations. Figure 3.1 illustrates an example of dataset images labeled using CVAT. In the example, grass is labeled with green color, roads are masked in yellow, buildings are marked in red, and miscellaneous objects are annotated in orange. And it is good to mention that CVAT offers a variety of image labeling formats such as COCO, YOLO, Segmentation Mask 1.1, Pascal VOC, etc. After the images of the dataset are uploaded to the CVAT server, each class is labeled with a polygon tool. Then the export format is chosen. When creating our model using the U-Net architecture, we chose Mask Segmentation 1.1 as the file export format. For example, if we work with Mask-RCNN, we have to choose the COCO extension, and YOLO export files have .txt format, and so on for other formats. For the segmentation mask in CVAT 1.1 export format, the original RGB pixel values are replaced by the values assigned to each category, as shown in Table 3.1.

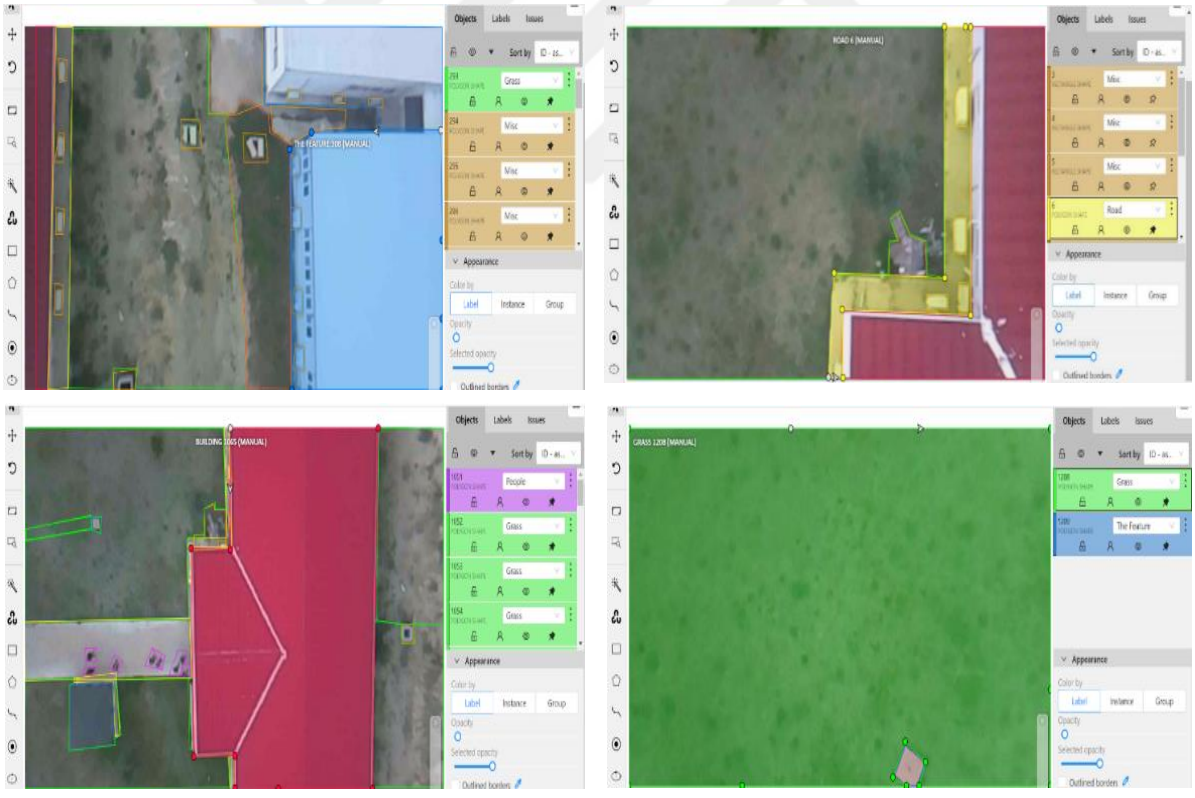


Figure 3.1. Examples Of Images Captured That Are Used in The

3.2. Image segmentation

The image classification process differs from image segmentation as it aims to identify the content within the image. In segmentation, objects are outlined pixel by pixel without drawing

a bounding box around them. Both classification and object detection processes are related to image segmentation. To achieve segmentation, the regions must be categorized, and then individual objects are detected and outlined with bounding boxes, segmenting the image pixel by pixel. In our scenario, the segmentation model is crucial for describing the landing point by ignoring the background of the dataset.

The U-Net architecture consists of an encoder and a decoder. The encoder, a typical convolutional neural network (CNN), progressively reduces the spatial resolution of the input image while capturing higher-level features. On the other hand, the decoder upsamples the low-resolution feature maps back to the original image size. The encoder follows a contracting path, involving repeated blocks of convolutional layers followed by downsampling operations like Max Pooling or Stride Convolution (Ronneberger et al., 2015). Each block increases the number of channels while reducing the spatial dimensions. The decoder, or path extension, consists of repeated blocks that upsample the feature maps while decreasing the number of channels (Zhou & Siddiquee, 2018). Skip connections combine the upsampled feature maps with the corresponding feature maps from the encoder, enabling the network to use both low-level and high-level features for accurate segmentation. By linking feature maps from the encoder to the corresponding upsampled maps in the decoder, the network recovers spatial information lost during downsampling and makes precise segmentation predictions (Zhang et al., 2021).

The U-Net architecture's final layer employs a 1x1 convolution to map features to the desired number of output channels, corresponding to the segmentation classes. The Softmax activation function is utilized to generate the segmentation mask.

The model is trained using a pixel-wise cross-entropy loss function. Adam optimization with backpropagation, a gradient descent technique, is used to optimize the model's parameters. The predicted segmentation and ground truth masks are fed to the network as input images to minimize the difference between the anticipated segmentation and the ground truth.

3.2.1. Feature Extraction

Image segmentation using the UNet architecture with a ResNet-34 backbone represents a symbiotic fusion of strengths for precise feature extraction and meticulous segmentation. The process entails several crucial steps. First, the ResNet-34 backbone, a deep convolutional neural network renowned for its aptitude in capturing intricate image details and semantics, serves as the primary feature extractor. It processes the input image through convolutional layers,

discerning hierarchical features across diverse scales. Concurrently, the UNet architecture enhances feature extraction by incorporating skip connections between the encoder (ResNet-34) and decoder components, facilitating the amalgamation of high-level semantic insights from the encoder with fine-grained spatial information from the decoder. This combined architecture is employed for semantic segmentation, where the input image undergoes feature extraction through the ResNet-34 backbone and is subsequently subjected to pixel-wise classification by the U-Net's decoder, resulting in a meticulously segmented image. The model is often pretrained on a substantial dataset for feature extraction and subsequently fine-tuned on a task-specific dataset to adapt to the target domain. The output is a segmented image with pixel-level labels, and post-processing techniques can be applied to refine the results. This approach finds application in diverse domains, including medical image segmentation, autonomous vehicle scene understanding, and satellite imagery object detection, offering accuracy across various contexts.

3.2.2. Training data

As per the known practice, the dataset was divided into two main sets. The first set, referred to as the training dataset, is a substantial subset of the original data and is utilized to train the model. The second set, known as the test dataset, is used to evaluate the accuracy of the model. The training and test groups were split in an 80:20 ratio, ensuring that the test data remains unseen by the model during training.

During the flight over ATU university campus, a quadcopter captured a total of 925 RGB images, each having a resolution of 1280x720 pixels. Some examples of these captured images are depicted in Figure 3.1. The RGB values corresponding to each segmentation class are presented in Table 3.1. Notably, each image is labeled with 9 segmentation classes.

Table 3. 1. RGB Values Corresponding to Each Segmentation Class

	Label	Color_RGB
1.	Background	0 , 0 , 0
2.	Building	250 , 50 , 83
3.	Grass	61 , 245 , 61
4.	Misc	204 , 153 , 51
5.	People	184 , 61 , 245
6.	Road	250 , 250 , 55
7.	The Feature	42 , 125 , 209
8.	Trees	255 , 106 , 77
9.	Vehicles	51, 221, 255

Consequently, the labeling process assigns a specific label to each pixel based on its content. Pixels associated with buildings are labeled as "building." Additionally, a "background" label is assigned to pixels that were disregarded during labeling. Pixels corresponding to flat plants or grass are labeled as "grass." "Road" labels are assigned to pixels representing paths, such as roads or sidewalks. Each type of vehicle, such as cars, buses, and motorcycles, is given a distinct "vehicle" label. "People" and "trees" labels are assigned to pixels representing individuals and trees in the dataset images, respectively.

For pixels representing objects used for testing purposes, the label "Feature" is applied. Lastly, the "Misc" label is assigned to pixels belonging to objects that do not fall into the same class as other objects, thus representing miscellaneous elements.

3.2.2.1. Tensorflow Format and Softmax Activation Function

Our models are designed for multiclass classification, meaning they can classify images into more than just two categories. In our dataset, we have a diverse set of images containing various elements such as grass, buildings, roads, people, and more. The goal of the model is to recognize the content of previously unseen images, i.e., images that were not part of its training data. To facilitate this multiclass classification, we utilize a hot encoding method for the input classes, as depicted in Table 3.2. Each category is represented independently, and there is no interdependence between different categories. This allows us to employ a generalized

independence method, ensuring that each region within an image is classified as a single category only.

In order to achieve a uniform encoding format, we export the labeling in Segmentation Mask 1.1 format, which aids in converting the image to the desired encoding scheme. This approach enables our model to effectively handle multiclass classification tasks and identify the contents of new and unseen images with accuracy.

Table 3. 2. Classes and Their Corresponding Indexes

Classes_Name	Indexes_of_Classes
Building	0
Grass	1
Road	2
Misc	3
The_Feature	4
Trees	5
Vehicles	6
People	7
Background	8

Consequently, our model is constructed using the SoftMax activation function, which produces a probability score vector as its output. This function plays a vital role in obtaining the final output vector of the neural network. (Equation 3.1 represents the mathematical expression of the SoftMax activation function).

$$\boldsymbol{SoftMax(Z)_i = \frac{e^{Z_i}}{\sum_{j=1}^{N} e^{Z_j}}} \qquad \textbf{3.1}$$

In the neural network, the vector Z represents the raw outputs, while (i) refers to the expected probability of the input for the test, corresponding to class I (the index of the class). N represents the total number of classes considered in the model. In this context, we use the value of e ≈2.718 (Nwankpa et al., 2018) for calculations.

The primary goal is to assign a value of one to the associated index of each pixel class in the last dimension, while setting all other index values to zero. To ensure clarity, the intended

output dimensions for this model are 512x512x9, where each pixel position has two values (representing the pixel position), and the last dimension is dedicated to class representation.

To achieve this, the segmentation mask is transformed into a null tensor with the desired output dimensions. The process involves iterating through each pixel and appropriately assigning the values to achieve the desired outcome. Figure 3.3 showcases some illustrative examples of the labeling process results for the training images along with their corresponding labels.

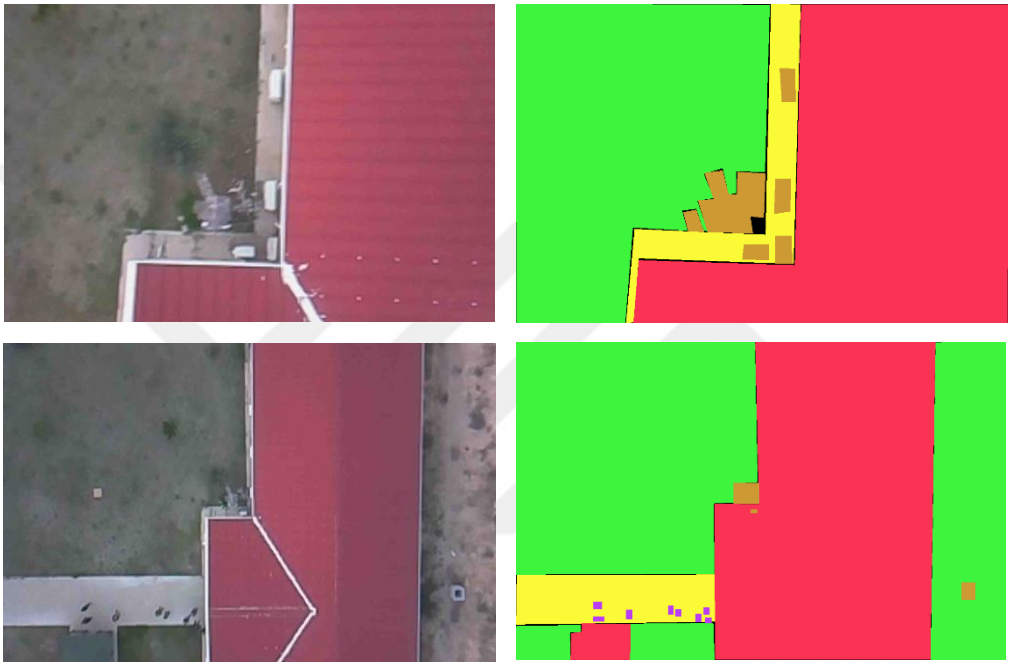


Figure 3. 2. Example of Some Training Images and Their Labels

3.2.2.2. Multiply The Data Set by Smaller Patches

Deep learning algorithms, especially during the training phase, often benefit from using small-dimensional datasets as they can lead to better results and higher performance. To achieve this, a Python library called Patchify comes into play, offering the capability to divide large-dimensional images into smaller patches. These patches are then stored as Numpy arrays, a scientific Python package, as part of the process (Kotaridis & Lazaridou, 2022).

With Patchify, the large images are effectively segmented into smaller, overlapping areas based on the specified patch size. These cropped patches are subsequently merged back into the original image. In the context of this particular model, the images were initially of size 1280x720x3 pixels. By applying the Patchify technique, they were resized to 512x512x3, effectively doubling the number of images in the dataset. Consequently, the dataset now

contains 1852 images, compared to the original 927 photos captured by the drones using their onboard camera. The same approach was also applied to the image label dataset, enhancing the overall dataset for training, and improving the model's performance.

3.2.2.3. Loss function

Selecting the right loss function to evaluate the accuracy of the model's performance is crucial, as is determining the appropriate neural network structure in the Deep Learning segmentation process (Sudre et al., 2017).

In this particular model, the loss function is calculated by considering the total loss, which accounts for the total between the dice loss and the focal loss, as depicted in Equation 3.2. This comprehensive loss function plays a pivotal role in guiding the model's training process, optimizing its performance, and effectively handling the segmentation task.

- **Dice Loss (Dice Coefficient Loss):** The Dice Loss is a measure of the similarity between two sets, commonly used in image segmentation tasks. It is defined as:

$$\text{Dice Loss} = \mathbf{1} - \left(\mathbf{2} \times |\boldsymbol{Intersection}| \Big/ |\boldsymbol{Ground\ Truth}| + |\boldsymbol{Predicted}| \right) \quad \mathbf{3.2}$$

Where:

$|Intersection|$: represents the cardinality (number of elements) of the intersection between the ground truth and predicted sets.

$|Ground\ Truth|$: represents the cardinality of the ground truth set.

$|Ground\ Truth|$: represents the cardinality of the predicted set.

The Dice Loss aims to maximize the similarity between the predicted and ground truth masks, with a lower value indicating better performance.

- **Focal Loss:** The Focal Loss is designed to address class imbalance and focus more on hard-to-classify examples during training. It is defined as:

$$\boldsymbol{Focal\ Loss} = -\boldsymbol{\alpha}\,(\mathbf{1} - \boldsymbol{P})^{\hat{\boldsymbol{\gamma}}} \times \mathbf{log}\,\boldsymbol{P} \quad \mathbf{3.3}$$

Where:

α : is a tunable parameter that controls the balance between easy and hard examples.

P : is the predicted probability that the example belongs to the correct class.

γ : is a tunable parameter that adjusts the rate at which the loss focuses on hard examples.

- **Total Loss:** It refers to the combined loss function used in the training of a neural network. this total loss function is a sum or a weighted combination of multiple individual loss functions as Equation 3.4. shown. The purpose of combining these loss functions is to provide a comprehensive measure of how well the neural network is performing across different aspects of the task.

$$\boldsymbol{Total_{Loss} = dice_{Loss} + (1 \times focal_{Loss})} \quad \textbf{3.4}$$

The Focal Loss effectively down-weights well-classified examples (where p is close to 1) and emphasizes the importance of hard examples (where p is closer to 0.5). This helps in cases where there is a significant class imbalance or when the dataset contains challenging examples. These loss functions are used during the training of neural networks for tasks like image segmentation, where the goal is to minimize the loss function to improve the accuracy and quality of segmentation results.

The Dice Similarity Coefficient (DSC), also known as the cube loss, is a widely adopted metric for assessing segmentation accuracy. It relies on dividing pixels into two groups: the real boundaries from the ground truth and the predicted pixel boundaries from the model's output. The DSC quantifies the degree of overlap between these two groups. When there is a substantial overlap, the DSC achieves its maximum value of 1, signifying a high level of agreement. Conversely, as the agreement diminishes, the DSC decreases towards its minimum value of zero, particularly in cases of significant mismatch.

To account for the variations in category weighting relative to their sizes, the cube loss incorporates a scaling factor. Equation 3.4 demonstrates the role of DSC in classifying each voxel as false-negative (FN), true (TP), and false-positive (FP). The use of DSC provides valuable insights into the segmentation performance, facilitating the model's optimization and improving its accuracy. The concept of DSC (Yeung et al., 2022) plays a crucial role in evaluating the segmentation results and guiding the training process.

$$\boldsymbol{\mathcal{L}_{DSC} = 1 - \mathrm{DSC}} \quad \textbf{3.5}$$

while

$$\mathbf{DSC} = \frac{2TP}{2TP+FP+FN} \quad \textbf{3.6}$$

Another fundamental aspect of loss computation is focal loss, which aims to tackle the class imbalance problem typically encountered in standard entropy loss. The focal loss reduces the emphasis on easy examples, allowing the model to focus more on learning from difficult examples. As a result, it serves as a variant of the binary entropy loss (Yeung et al., 2022).

Equation No. 3.7 forms the foundation of this model, as it is specifically designed for multi-class segmentation tasks. The focal loss plays a significant role in improving the model's ability to handle imbalanced class distributions and enhances its performance in challenging segmentation scenarios. By incorporating the focal loss into the loss computation, the model can better address the difficulties associated with class imbalance, leading to more accurate and reliable segmentation results.

$$\mathcal{L}_{CF} = \alpha\,(1 - (\mathrm{P}_{t,c}))^{\gamma}.\ \mathcal{L}_{CCE} \quad \textbf{3.7}$$

$$\mathcal{L}_{CCE}(y, p) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}.\log(\mathrm{P}_{i,c}) \quad \textbf{3.8}$$

Where is the categorical cross-entropy loss, α is a class weights vector, $\mathrm{P}_{t,c}$s the matrix for each class of true probabilities, N is many classes, i repeat for all pixels, c iterate classes, $y_{i,c}$ is a basic truth label that uses a one-hot encoding planner and $\mathrm{P}_{i,c}$I indicate to each cs and its expected matrix of values and utilizing the focusing parameter γ (where γ>=0) as a modulation factor to diminish the impact of accurately classified samples within the loss function. When γ=0, the Focal Loss aligns with the Binary Cross Entropy Loss.

3.2.2.4. Optimization Algorithm

An optimization algorithm is utilized to address optimization problems by enhancing a specific quantity under certain conditions (Briggs, 1966). In the context of Deep Learning, optimization algorithms play a crucial role in updating network weights and minimizing the loss function during the training process. The choice of an appropriate optimization algorithm is of utmost importance since it significantly impacts the quality of the model's training, which can span days, weeks, or even longer.

Considering the available data, the Adaptive Moment Estimation (Adam) optimization algorithm was deemed the most suitable. Adam is well-suited for handling large parameter sets

and is particularly effective for large datasets. This algorithm delivers impressive computational results while requiring minimal memory usage. Additionally, it excels when applied to dynamic or non-fixed targets (Alom, 2021). By utilizing the Adam algorithm, the training process for the Deep Learning model can be optimized to achieve better performance and more efficient convergence.

3.2.2.5. Adam Optimization

The core concept behind Adam optimization is to dynamically adjust the learning rate for each parameter based on the estimates of the first and second moments of the gradients. It achieves this by maintaining running averages of past gradients (first moment) and squared gradients (second moment) for each parameter. Adam optimization incorporates and refines the mechanisms of AdaGrad and RMSProp (Norouzi & Ebrahimi, 2019).

AdaGrad, an adaptive gradient algorithm, maintains a specific learning rate for each parameter, which proves beneficial in various problems, particularly in computer vision and scenarios involving sporadic gradients. On the other hand, RMSProp demonstrates its effectiveness in non-stationary problems and online settings. It adapts the learning rates of parameters based on the average change in the speed of recent gradients.

Notably, the Adam optimizer utilizes non-centered variance, distinguishing it from the RMSProp mechanism, which relies on the average of the first moment to adapt the learning rates of parameters. The mathematical Equation 3.9 illustrates how the Adam optimizer optimizes the learning rates to achieve better performance during the training process. By incorporating the best of both AdaGrad and RMSProp, Adam optimization emerges as a powerful and effective optimization algorithm in various deep learning applications.

$$\boldsymbol{\omega_{t+1} = \omega_t - \widehat{m_t}\left(\frac{\eta}{\sqrt{\widehat{v_t}}+\varepsilon}\right)} \qquad \textbf{3.9}$$

While $\widehat{m_t}$ bias-corrected weight parameters also the $\widehat{v_t}$, ε is a very small positive constant to ignoring the error of division by 0 its value is almost 10^{-8}, η is a learning rate (0.001) and ω_t is a weight at time t, and ω_{t+1} is a weight at time $t+1$.

4. Landing Site Selection

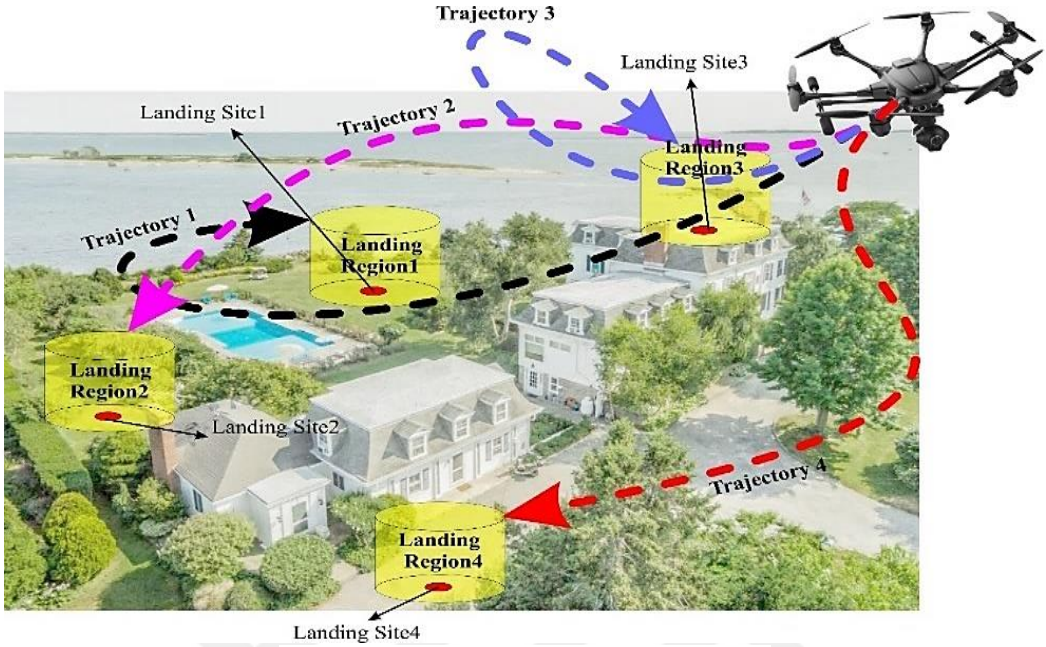


Figure 4. 1. Options of Landing Sites

The objective of this study is to develop a landing system framework exclusively reliant on processing images acquired from the UAV's onboard camera. The process involves utilizing the U-Net and ResNet34 backbone detection algorithm to identify objects within the images. Subsequently, semantic image segmentation techniques are explored to ascertain safe landing zones, exemplifying the effectiveness of lightweight models that reduce the dependency on numerous sensors. As a result, multiple potential safe landing sites may be identified, as illustrated in Figure 4.1.

4.1. Safest landing side

Selecting the appropriate landing site is a critical step in emergency situations, as it directly impacts the feasibility of the landing. For instance, the distance between the drone and the landing site, along with a model to calculate the total energy consumption while following a path, are crucial factors in determining whether the drone can safely reach its destination. Moreover, maintaining a safe distance from obstacles along the path is essential to avoid collisions in the event of a complete power failure.

This research aims to identify the optimal landing site based on two main criteria. Firstly, the safest landing site is determined by maximizing the distance from obstacles. Secondly, the most energy-efficient trajectory to the landing site is sought, considering the available options. Consequently, different criteria play a role in influencing the selection of the landing site, as outlined in Figure 4.2.

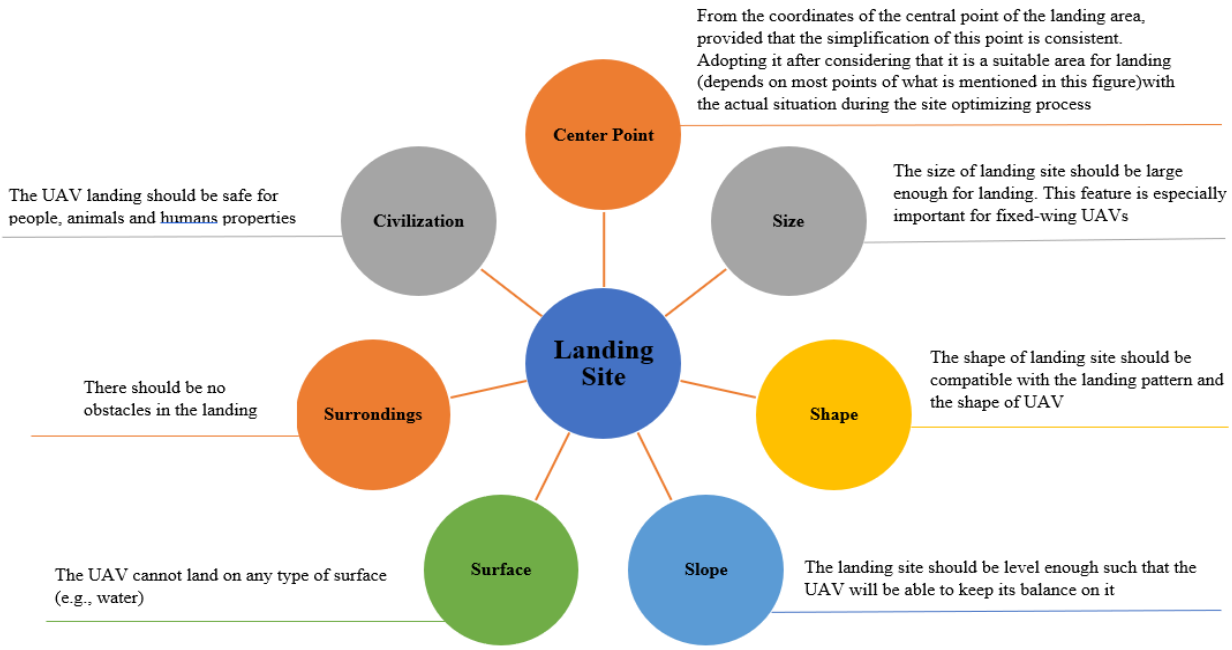


Figure 4.2. Safest Landing Parameters

4.1.1. Largest Clearance from The Obstacle (Surrounding)

In this case, all obstacles are considered equally important in terms of collision risk, making it essential to maintain an equal distance from them whenever feasible. To identify such points on a map, the Generalized Voronoi Diagram (GVD) proves to be one of the most effective methods. Since the landing site's z-component is always zero, representing the ground level, the search is confined to the x-y plane of the map.

To facilitate this thesis, the map is discretized with a fixed step size, typically set at 1 meter in this work. For larger maps or scaled representations, the step size can be adjusted accordingly to reduce computation time.

To determine the optimal landing site using the GVD, a mesh of obstacle-free paths, representing the edges of the GVD, is generated in the x-y plane of the map. For each point

within this network, a cost function J is calculated based on two factors: the distance from obstacles and the distance to the vehicle. The goal is to find a point with the lowest cost J, ensuring a safe and efficient landing site.

$$J_{(r,d)} = \frac{a}{r} + bd \qquad 4.1$$

Here, a and b represent two adjustable weights for calculating the clearance from obstacles *(r)* and *(d)* the distance from the vehicle respectively. Finally, by calculating Equation 4.1 for all points in GVD, the point with minimum cost can be selected as the best landing spot. If multiple points are returned, priority is given to the one with the minimum distance from the vehicle. Figure 4.3 shows some obstacles at different points.

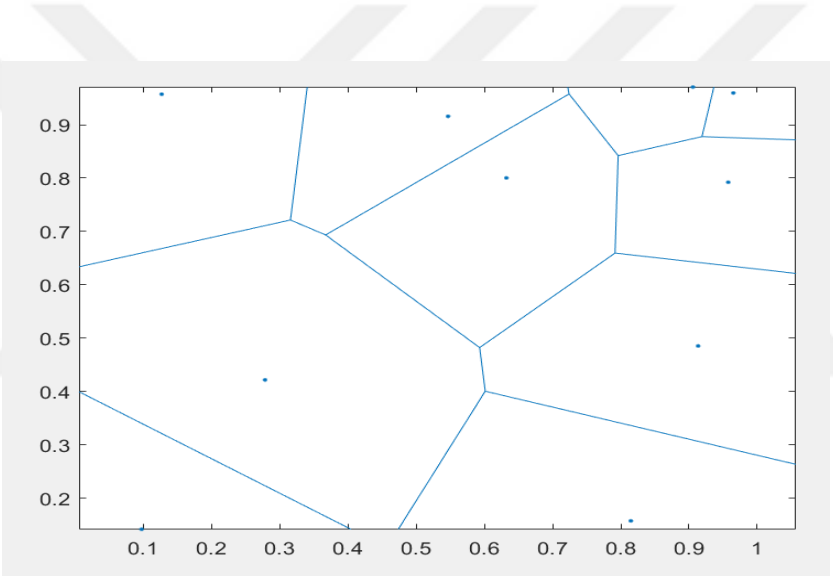


Figure 4. 1. The Collection of All
Voronoi Polygons for Every Point in the Set.

4.1.2. Size

To choose the appropriate size of the landing zone, the landing site should provide enough space for the drone to land comfortably without the risk of overshooting or colliding with nearby objects.

The appropriate landing zone size for a quadcopter can depend on several factors, including the size and type of the quadcopter, its flight characteristics, and the purpose of the landing. However, we will focus on the takeoff, landing, and emergency landing areas (Mung et al., 2022).

Takeoff and landing area: for safe takeoff and landing, it is generally recommended to have a clear, flat area with a diameter at least twice the rotor span of the quadcopter. This allows adequate clearance and reduces the risk of the rotors striking obstacles during these critical phases of flight.

Emergency landing area: It is advisable to have a larger landing area available for emergencies. Some experts recommend a diameter of at least four times the rotor span to have sufficient space for unexpected situations or errors during landing.

4.1.3. Shape

Depending on the unique requirements and conditions, the shape of the quadcopter landing site may change. The following typical factors should be considered while choosing the geometry of a quadcopter landing site:

- Flat and Level Surface: For the quadcopter to land securely, the surface must be flat and level. This promotes stability during takeoff and landing (Tsintotas et al., 2021).
- Non-Slip Surface: Having a non-slip surface at the landing location is advantageous for ensuring a secure landing (Mat, n.d.). This helps, especially in slick or muddy circumstances, prevent the quadcopter from slipping or skidding during landing or takeoff.

While the design of a quadcopter landing site is not predetermined, the most typical geometries are rectangular or circular, depending on the available area and the operator's preferences. In the end, the landing site's design should put safety, usability, and the unique operational needs of the quadcopter first.

4.1.4. Slope

In this thesis, we did not rely on the slope method for identifying safe landing areas. However, as part of a preliminary experiment, the slope method involves partitioning a scanned area into smaller squares and assigning potential hazard heights to these squares (Shah Alam & Oluoch, 2021). A gradient of colors is utilized to represent the heights of the squares, thereby illustrating obstacles and potential landing zones in a comprehensive view. Through the Safe Landing Area Determination (SLAD) algorithm, optimal landing sites are identified by incorporating factors like surface slope, obstacle height, and hazard proximity constraints (Ayhan et al., 2019).

To ensure safe and dependable operations, it is generally recommended to choose a flat and level surface for a quadcopter landing site. However, certain situations may allow for some flexibility where a gentle slope could be deemed acceptable. When evaluating the slope of a quadcopter landing site, consider the following resources and ideas to make an informed decision:

- Federal Aviation Administration (FAA): The FAA does not establish a specific slope requirement for landing locations, but it does issue advice for safe drone operations (Aviation, 2016).
- Flight Characteristics: Slopes can have an impact on a quadcopter's flight characteristics. A steep slope can make the aircraft unstable during takeoff or landing, which could result in tipping or loss of control.
- Personal Experience and Pilot Skill: Skilled drone pilots with experience may be able to land safely on gentle slopes by modifying their approach and inputs. A flat, level landing spot is still preferred whenever possible because this needs a great level of ability and experience. maintaining a space that is open and free of obstructions is crucial during takeoff and landing.

4.1.5. Surface

A landing site's suitability for a quadcopter depends on several criteria, including:

- Flat and Clear Surface: The landing area should preferably be flat and free of any obstructions that can hinder the quadcopter's landing or takeoff, such as trees, buildings, power lines, or other structures.
- Surface Texture: The designated landing area must possess a suitable surface texture that provides adequate traction for the quadcopter's landing gear. Typically, flat and smooth surfaces like concrete or asphalt are preferred over rough or uneven terrains. Interestingly, we did not rely on the surface texture method. However, successful experiments have shown that surface reconstruction involves generating a mesh of interconnected points to depict a three-dimensional surface, often assumed to be stable and unchanging. In the realm of computer vision, the objective is to reconstruct objects or environments using sequences of images. In the context of UAVs, this research centers on estimating depth to evaluate the safety of landing areas. The study introduces two approaches rooted in helicopter movements: one with a consistent altitude and

translational motion, and the other simulating vertical landing. Both strategies employ vision systems based on distinct features to analyze landing areas (Cesetti et al., 2010).

For the first approach, an optical flow-based depth perception system is utilized for safe autonomous landing. The method relies on stable image features for large motions, yielding a sparse set of confident depth estimates. If the surface is deemed stable, it's considered suitable for autonomous landing (Mancini et al., 2013). The procedure involves capturing an image sequence of the landing area, computing optical flow using SIFT features, recovering depth information, and using a binary threshold classifier to determine if the surface's depth is consistent, indicating its usability as a landing area.

- Absence of Loose Debris: The landing spot needs to be devoid of any loose objects that the quadcopter's propellers could harm. Examples of such objects are gravel, sand, and loose plants.
- Environment: Consider factors including visibility, wind direction and speed, and potential weather dangers like rain and snow. It's crucial to maintain clear skies and safe landing conditions.

4.1.6. Civilization

The particular location and context would determine if a quadcopter landing site was suitable for human habitation or civilization. In general, it's a good idea to take the following things into account:

the population density of the area where the quadcopter will be landed. Landing in crowded metropolitan areas can be dangerous because of the presence of buildings, people walking, and moving automobiles. To reduce the chance of mishaps, it's crucial to put safety first and make sure the landing place is far from populous areas.

4.1.7. Proper Landing Area

We decided to designate the center area of each object in the image as the landing area after taking the nature of the area depicted in the photos into account and consulting the classification and segmentation stages as a guide. Following the object detection, each item's bounding box coordinates are recovered, as illustrated in Figure 4.4. As shown, the red rectangle first ringed the grass area before being followed by the second rectangle around the feature region. The

center of the feature and the grassy area were then determined, together with their label number. The bounding box represents the region of the image containing the object.

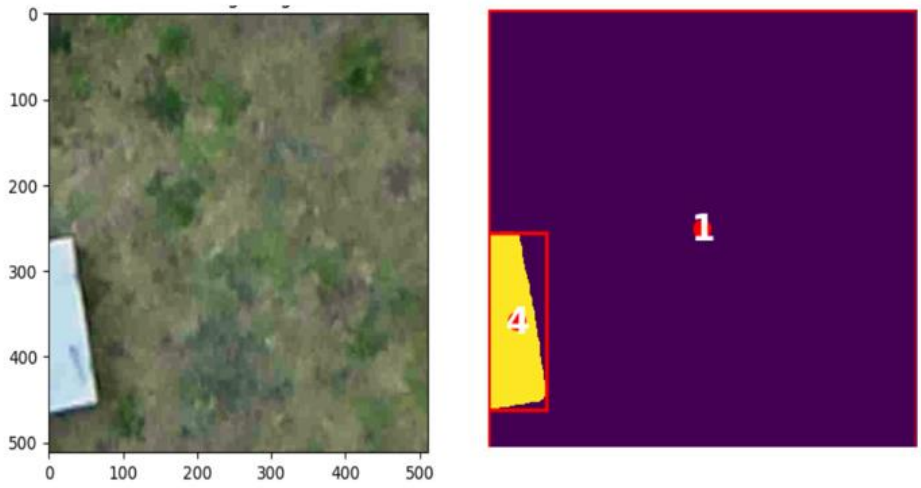


Figure 4.4. Centre Point and the Box Surrounding the Landing Spot

To calculate the midpoint coordinates of each object's bounding box, the center point is determined by adding half of the width to the x-coordinate of the upper-left corner and half of the height to the y-coordinate of the upper-left corner. The bounding box coordinates consist of the top-left corner (x, y) and the width and height values. Initially, an image (test_img) and its corresponding ground truth label (ground_truth) are randomly selected from the X_test and y_test_argmax datasets. Subsequently, the dimensions of the image are expanded using the np.expand_dims function from the NumPy library. This function enables the addition of an extra axis at a specified position, effectively expanding the array's dimensions in preparation for prediction.

The trained model is then utilized to predict the label of the test image, with the prediction being saved in the 'prediction' variable. The predicted image is obtained by using np.argmax, another function provided by the NumPy library. This function returns the indices of the maximum values along a specified axis in an array. In this case, it is employed to locate the index with the highest value along the third axis. The resulting image is stored in the 'predicted_img' variable.

5. Path Planning

This step involves guiding the UAV to the landing area. Several approaches can be used to draw the path and select the best landing location using artificial intelligence. As we mentioned in this thesis, this process depends on processing the images captured by the quadcopter's camera during image processing.

5.1. Introduction

A distributed multiagent path planning technique for quadrotors in dynamic situations is studied in reference (T. Huang et al., 2021). According to Dag et al. (2022) , a framework for energy-based path planning is utilized to increase the flight endurance of a solar-powered UAV. To create a viable path for a UAV, multi-objective path planning is provided (Kim & Lee, 2018), where the proposed algorithm takes safety into account. Belge et al. (2021) introduce a quadrotor UAV in an unknown unstructured outdoor environment as a path planning approach based on an elliptical tangent model to lessen computing load. Yüzgeç and Üçgün (2021) investigate a ground feature-oriented strategy to produce an appropriate path for UAV mapping. Two path planning algorithms are developed, one of which is based on successive convex approximation and the exact penalty method, while the other of which uses a heuristic methodology (Wei & Xu, 2021). Additionally, Luo et al. (2022) offer an enhanced A-star algorithm to create a path for a UAV to follow when tracking a target. By Huang et al. (2021), a strategy for collaboratively attacking many targets in a static environment is proposed using several UAVs and Voronoi diagrams. To achieve 3D path planning of a UAV, an enhanced rapidly exploring random tree (RRT) technique is introduced by (Huang et al., 2021). In some situations, such as those involving mobile robots (Kovács et al., 2016; Orozco-Rosas et al., 2019). Two drawbacks of the classic APF (TAPF) are the local minimum and the unachievable target. The repulsive potential function of the TAPF is swapped out for a Gaussian function to address these issues (Huang et al., 2022). When an obstruction is on the line connecting the current position and the target position, the UAV could still experience a local minimum even when using the improved APF (Liu et al., 2020).

5.2. The Algorithm

A prominent and commonly used pathfinding method in computer science (Yan, 2023) is the A* (pronounced "A-star") (Hart et al., 1968) that is used in this thesis. Finding the shortest path between two nodes in a network effectively is done using an algorithm that does informed searches (Cormen et al., 2009). The A* method considers the cost of traveling to a node from the starting node as well as an estimate of the cost to go to the goal node, known as the heuristic function.

The heuristic function that is selected has a significant impact on the A* algorithm's effectiveness and efficiency. Without overestimating it, the heuristic function should offer a lower bound estimate of the cost to the objective node. The Manhattan distance or Euclidean distance between the current node and the desired node is the most widely utilized heuristic.

Dijkstra's algorithm, which always finds the shortest path, and greedy algorithms, which can be faster but may not always discover the best path, are combined into the A* algorithm. It is widely utilized in many different applications, including robots, video games, and route planning (Russell & Norvig, 2010).

The following stages were performed by the A* algorithm:

- A locked list and an open list were made from nothing. Nodes that have been discovered but have not yet been looked into can be found on both the open list and the closed list.
- After that, the original node was added to the open list and given a cost to reach *(g-score)* of 0.
- Even if the open list is not completely empty, the following actions will be repeated:
    a) The node with the lowest *f*-score *($f = g + h$)*, where g is the node's trip expense and h are a heuristic estimation of the goal's journey expense, should be chosen from the open list.
    b) Changed the position of the selected node from the open to the closed list.
    c) The selected node was examined to see if it is the goal node. If so, the route has been discovered.
    d) It will create the specified node's surrounding nodes.
    e) Calculated the g-score (cost to reach from the start node) and h-score (heuristic estimate of the cost to the target) for each surrounding node.

f) The surrounding node will be disregarded if it is already in the closed list or has a higher g-score. Alternatively, it will have its g-score updated and be added to the open list.

g) A node will update its g-score if a node with a higher g-score is already in the open list.

h) Recorded the chosen node as the surrounding node's parent.

- There is no path between the start and goal nodes if the open list is empty.
- When the goal node is attained, it reconstructs the shortest path by following the parent pointers that lead from the goal node back to the start node.

Within the A* algorithm, the minimum distance cost function is typically structured by integrating two core components:

- The Cost to Reach a Node *(g(n)*): This component signifies the genuine cost associated with reaching a specific node, with the node of interest often designated as the center of the landing area. In our specific case, this corresponds to the cost of traversing from an initial node, which we consider as the center of the image captured by the drone's camera. The cost calculation can encompass various factors such as distance or time. In our context, it is interpreted as the distance between the center of the captured image and the center of the rectangle that surrounds the landing area.

- The Heuristic Estimate to Reach the Goal *(h(n)):* This constitutes an estimation of the cost to travel from the current node to the ultimate goal node, as previously elucidated. The heuristic component is problem-specific and confers a heuristic value to each node, grounded in the anticipated cost of attaining the final goal.

The minimum distance cost function *(f(n))* is subsequently defined as:

$$\boldsymbol{f(n) = g(n) + h(n)} \qquad \textbf{5.1}$$

Nodes, within this framework, are identified as discrete entities within the search space or graph, signifying distinct states or locations. Throughout the *A** algorithm, nodes are systematically explored, with each node evaluated in relation to the minimum distance cost

function *(f(n))*. Node expansion is contingent upon their associated *f(n)* values, with nodes boasting lower *f(n*) values accorded priority, as they represent more promising potential paths.

The overarching objective of the *A** algorithm is to identify the path from the initial node to the final goal node that incurs the least total cost. This consideration integrates both the actual cost incurred *(g(n))* and the heuristic estimate of the cost to attain the goal *(h(n))*. This fusion of factors renders the *A** algorithm proficient in identifying optimal paths across diverse search and optimization scenarios.

5.2.1. Algorithm parameter

- **batch_size**

  After conducting a series of experiments, monitoring the training process, and considering various factors, the optimal batch size for the UNet with ResNet-34 model was determined to be 8. Several key considerations influenced this choice:

  1. Dataset Size: Given that our dataset is relatively small, selecting a smaller batch size is a recommended practice to mitigate the risk of overfitting.
  2. Generalization: Smaller batch sizes were observed to potentially enhance model generalization. This is attributed to the model being exposed to a broader diversity of data in each training epoch.
  3. Training Speed: Smaller batch sizes facilitated faster iterations during training, which proved advantageous for expeditiously experimenting with different configurations.
  4. GPU Memory: Due to limited GPU memory, a smaller batch size was selected to prevent out-of-memory errors and ensure efficient model training.

  Taking these factors into account, a batch size of 8 was chosen as it struck a balance between mitigating overfitting, promoting generalization, and efficiently utilizing available computational resources.

- **Epochs**

The number of epochs in a Python implementation serves as a crucial hyperparameter, dictating the frequency with which the entire training dataset is subjected to forward and backward passes through the neural network during the training process. Each epoch corresponds to a full cycle through the training data. The determination of the appropriate number of epochs hinges on a myriad of factors, encompassing the model's intricacy, dataset size, and the specific problem under consideration. In this context, it was ascertained that 30 epochs constituted the optimal choice for this task, with the following factors being instrumental in this decision:

1. Model Convergence: Notably, after 30 epochs, the training model commenced its convergence phase. Convergence signifies a state where the loss on the training data stabilizes, and the model's performance reaches a plateau.
2. Early Stopping: The incorporation of early stopping was instrumental in averting overfitting. This entailed vigilant monitoring of the validation loss, prompting the cessation of training when signs of loss escalation or stagnation emerged. This safeguarded the model from learning noise in the training data.
3. Dataset Size: Due to the relatively modest dataset size, convergence was achieved at an expedited pace.
4. Learning Rate Schedule: The utilization of a fixed learning rate, as opposed to learning rate annealing, influenced the number of required epochs.
5. Validation Performance: The pivotal aspect of monitoring validation performance played a pivotal role in arriving at the epoch count of 30. This ensured that the model exhibited robust generalization to previously unseen data.

It is essential to recognize that the number of epochs can fluctuate significantly, spanning from a few to potentially hundreds, contingent upon the specific problem at hand and the variables enumerated above.

- **Verbose**

  In various Python libraries, notably TensorFlow and Keras, the 'verbose' parameter assumes a pivotal role in governing the quantity of information presented throughout the training process of machine learning models, including those encompassing a UNet architecture coupled with a ResNet-34 backbone. The 'verbose' parameter is designed to accommodate an integer value, thereby encapsulating the desired degree of verbosity. Conventionally, the following values are ubiquitously employed:

  1. verbose=0: This setting engenders a state where no informational output is showcased during the training process.
  2. verbose=1: At this level, progress bars or log entries materialize for each epoch, furnishing insights into the ongoing training progression.
  3. verbose=2: Leveraging this option elicits the manifestation of progress bars or log entries for each epoch, providing not only an overview of the training advancement but also supplementing details such as loss and metric assessments.
  4. verbose=3 or higher: At these elevated levels, there is the prospect of witnessing more intricate debugging information, contingent upon the particular library and its underlying implementation.

  In the context of this task, 'verbose=1' was thoughtfully selected to regulate the depth of information conveyed throughout the training process, ensuring that it aligns harmoniously with the requisites of the endeavour.

- **Weights**

  The term "weights" in the context of neural networks refers to the learnable parameters that play a central role in model learning and prediction. These weights are numeric values that underlie the connections between individual neurons within the network. In the specific context of convolutional neural networks (CNNs) like ResNet-34 and UNet, these weights are particularly

associated with the convolutional kernels or filters responsible for extracting distinctive features from input data.

- Regarding initialization and training, the weights initially receive specific values, such as 0.1666 for all objects, as part of the model's initialization process. Subsequently, during the training phase, these weights undergo iterative updates facilitated by optimization algorithms like Adam. These updates are orchestrated to minimize a predefined loss function, and they enable the model to progressively adapt to the nuances of the training data, thereby acquiring the knowledge necessary for various tasks, such as image segmentation or classification.

It's essential to underscore that these weights constitute a fundamental and pivotal component of a neural network, as they encapsulate the acquired knowledge from the training data. Consequently, these weights fundamentally govern the network's proficiency in making predictions and executing tasks intrinsic to its designated role.

- **Random state**

The random state parameter is utilized during the dataset splitting process to manage randomness in operations. It serves the purpose of maintaining result consistency across multiple code executions by establishing a fixed seed for the random number generator. In practical terms, 42 was specifying a particular random state value and it ensured that identical random outcomes are obtained when the code is rerun. This attribute proves valuable for debugging and guaranteeing result.

5.3. Path Planning

In order to discover a path from a start node to a target node in an image, we built an *A** search algorithm implementation (astar_search). The algorithm also includes:

1. The Euclidean distance between two nodes is calculated by the heuristic function (heuristic).
2. For a given node, the generate_neighbors method creates its neighbors.

Open set, a dictionary to store the previous node in the shortest path (came_from), a dictionary to store the g-score (cost from the start node to the current node), and a dictionary to store the f-score (estimated total cost from the start node to the target node) are among the data structures that are initialized. The *A** algorithm will then be executed until either the target node is reached, or the open set is empty.

The path is reconstructed by traversing backward from the target node to the start node using the came_from dictionary. Next, the center coordinates of the test image are determined, and if a path is found, it is returned. Otherwise, none is returned. To label the anticipated image regions, the skimage.measure.label function is utilized. Additionally, the skimage.measure.regionprops function, provided by the scikit-image library (skimage), is employed to compute various properties of labeled regions in the image. This function is commonly used in image processing and analysis tasks, particularly in computer vision. It helps extract the characteristics of regions. By locating the region with the label "The_Feature," the target region can be identified. The program checks if the desired area can be located. If not already done, the test picture, ground truth label, *A** path (if available), and target region are plotted. Figure 5.1 depicts the flowchart illustrating the process of drawing the path after determining the landing spot.

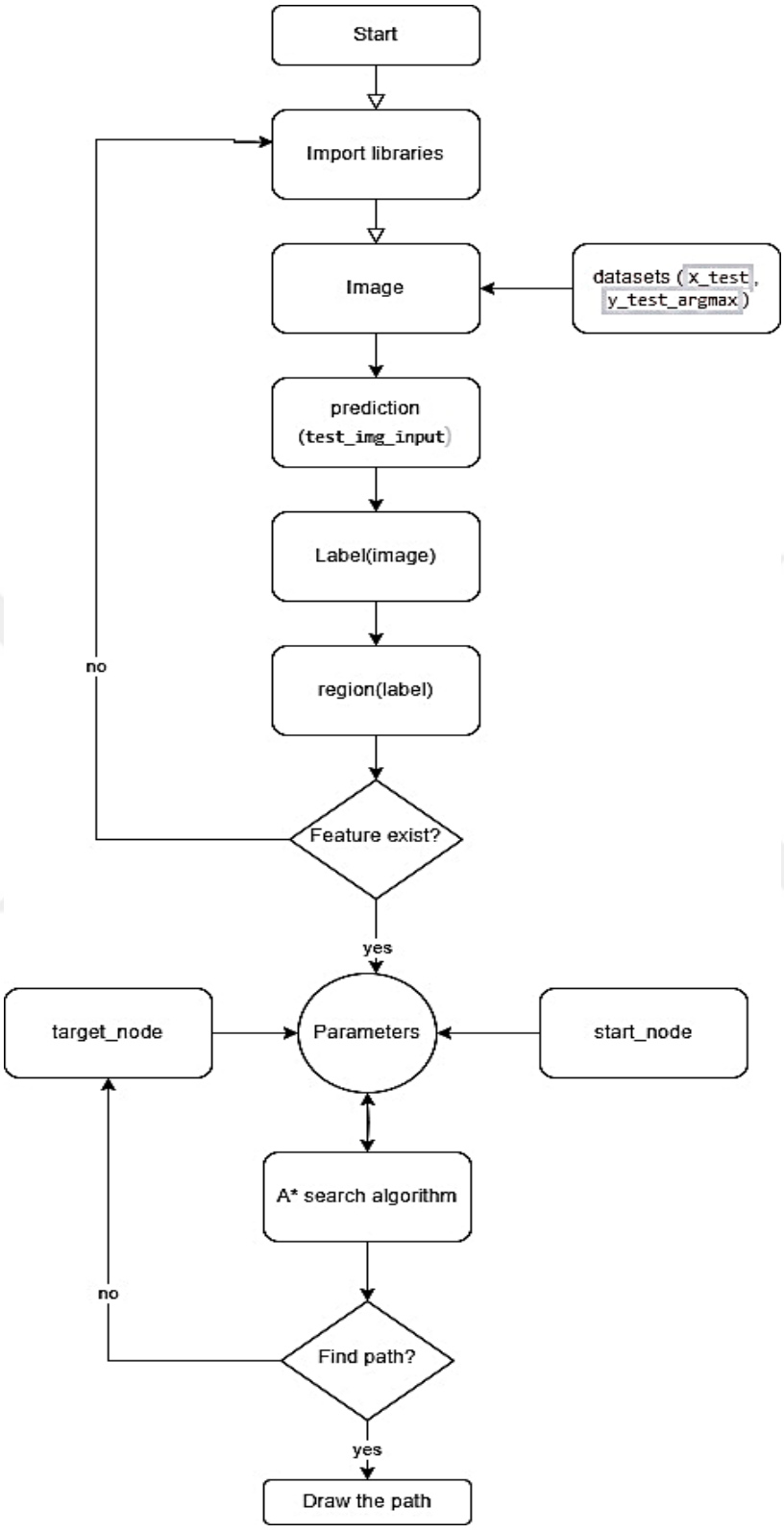


Figure 5. 1. Flowchart Of Path Planning

6. Applying Different Kinds of CNN Architectures

Testing various CNN architectures was a crucial step in our pursuit of accurate model creation. One prominent approach we explored was utilizing the Mask-RCNN function for classification and segmentation tasks on our dataset. While this method showed promise, we encountered challenges in obtaining satisfactory results. These difficulties were attributed to the diverse nature of our dataset and the content present in the images.

To address these issues, we undertook an extensive investigation of different scenarios. One approach involved adjusting the size of our dataset, trying both doubling and halving its size to observe any impact on the results. Despite our efforts, the desired outcomes remained elusive. In parallel, we decided to experiment with another CNN architecture known as the U-Net model. This alternative approach aimed to replicate the data augmentation process performed by the Mask-RCNN. By employing the U-Net model, we hoped to improve the performance and achieve better results.

The process of exploring multiple CNN architectures and analyzing their performance was an essential aspect of our research, ultimately driving us towards finding the most suitable solution for our unique dataset.

6.1. Mask-RCNN

To achieve image segmentation through Mask-RCNN and object detection using the Fast-RCNN architecture, we first embarked on a classification process for our dataset. This entailed labeling 925 images using the Computer Vision Annotation Tool (CVAT) online application, an Intel tool dedicated to this purpose. The labeled dataset played a crucial role in training the algorithm, enabling it to gain a deeper understanding of real-world objects under various conditions.

Our classification process involved assigning labels to objects within the images. The identified categories were as follows: Features, Misc, Vehicles, Trees, People, Road, Buildings, and Grass. The "Misc" label encompassed objects that did not fit into any other specific category, while "Features" referred to cardboard objects with a plus sign on them. We utilized the Polygon label (hexagon symbol) within the CVAT online app, which facilitated the drawing of masks around the objects, streamlining the labeling process. The labeled images were then exported

in the COCO 0.1 format with a ".json" extension, meeting the requirements of the Mask-RCNN algorithm.

Convolutional neural networks (CNN) played a central role in our work, as they are designed to process pixel data and excel in image recognition and processing tasks. Our chosen architecture, Mask-RCNN, evolved from Faster R-CNN, which had two outputs per candidate object: a class label and a bounding-box offset. Mask-RCNN introduced a third branch dedicated to predicting object masks (Region of Interest), allowing for more precise spatial layout extraction. This architecture exhibited several advantages, including ease of training, simplicity, and the ability to generalize to other tasks, such as estimating human positions in the same frame, all while maintaining efficiency and speed.

To optimize the performance of the MASK-RCNN algorithm, several parameters were crucial. The learning rate (LR), which determines the rate of minimization in the cost function during each iteration, required careful selection. It should strike a balance between being too large, causing the algorithm to miss the optimal solution, and being too low, leading to prolonged convergence. In our algorithm, we set the LR value to 0.00025. Another vital parameter was the number of iterations (MAX_ITER), which we controlled based on the mean Average Precision (mAP) value. Adjustments to MAX_ITER were made to prevent overfitting and achieve optimal results. The mAP values depended on the Intersection over Union (IoU), a measure of overlap between two boundaries.

In the training process, an epoch represented a single iteration where all batches were processed in both forward and backward propagation. Each epoch played a significant role in the algorithm's training stages, allowing it to make progress towards achieving the desired segmentation and object detection outcomes.

6.1.1. Result of MASK-RCNN

After the algorithm finished training. The testing process started, in every image, it made an expected percentage for each object. To clarify, Figure 6.1 shows the percentage of each object as Table 6.1 explains it.

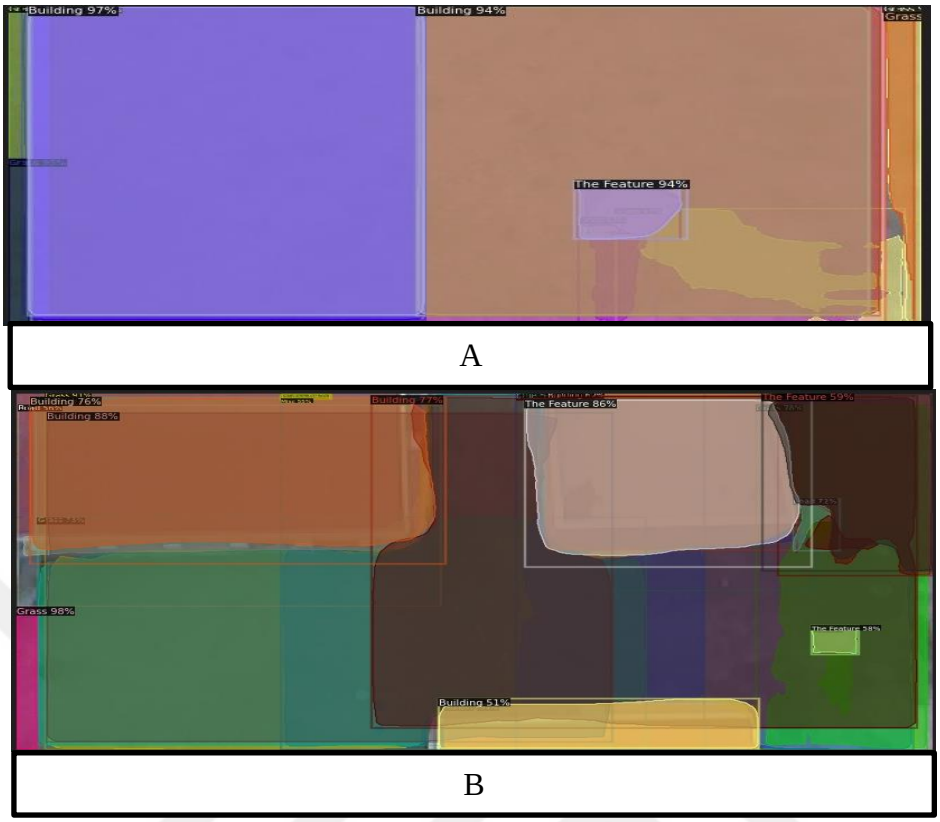


Figure 6. 1. Example of MASK-RCNN Expectation Success Percentage

Table 6 1. Some Classes and their Predictions of Success Percentage

Image	Classes	Success percentage
A	Building (1) Building (2)	**%97** **%94**
	Grass	**%95**
	The feature	**%94**
B	Building (1) Building (2) Building (3)	**%76** **%88** **%51**
	Grass (1) Grass (2)	**%73** **%98**
	The feature (1) The feature (2)	**%86** **%59**

We obtained success rate results for each object individually from a dataset comprising approximately 186 images. Table 6.2 displays the average success rate achieved for each object.

Table 6 2. The Average of Success Rate for Each Object

	objects	Object detection %	Segmentation %
1	Building	64.91	67.00
2	Grass	63.92	60.40
3	Feature	60.26	68.29
4	Road	36.03	17.43
5	Misc	19.84	18.89
6	People	8.67	5.10
7	Vehicles	0.0	0.0
8	Trees	Nan	Nan

Conversely, the algorithm's performance on the Training dataset provided insights into its successful training process. Figure 6.2 visually illustrates the variations in the appearance of results during the object detection stages.

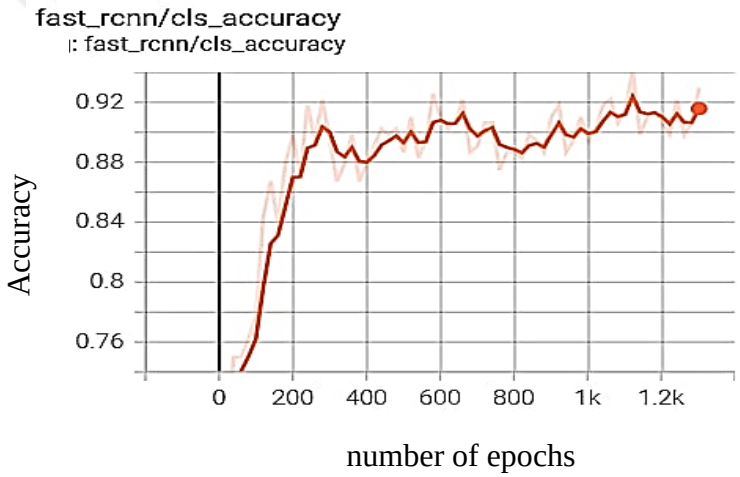


Figure 6. 2. Object Detection Variance

Figure 6.2 demonstrates that the accuracy of object detection on the Training dataset ranged from 72% to 92%. These results indicate the algorithm's effective performance, with accuracy values steadily increasing up to 92%.

In the segmentation process, we observed a distinct accuracy rate for objects within the images, as depicted in Figure 6.3.

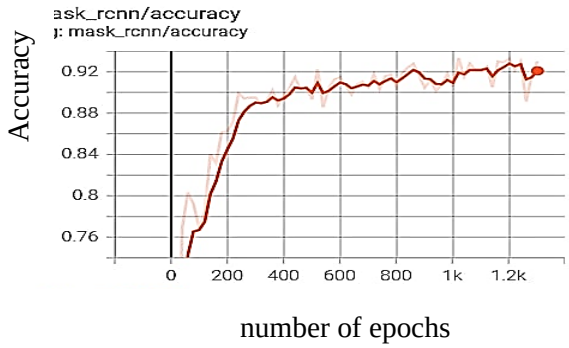


Figure 6. 3. Segmentation Variance Results

The accuracy rate during the segmentation stage on the Training dataset ranged from 72% to 92%. This increase in accuracy indicates an improvement as the training process advanced.

Furthermore, the accuracy based on the loss function for the Training dataset can be calculated, as illustrated in Figure 6.4. This metric provides valuable insights into the algorithm's performance and optimization during the training process.

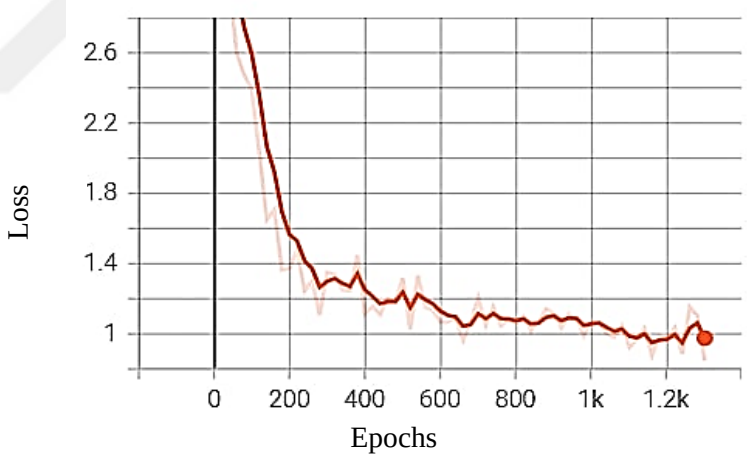


Figure 6.4. Total Loss

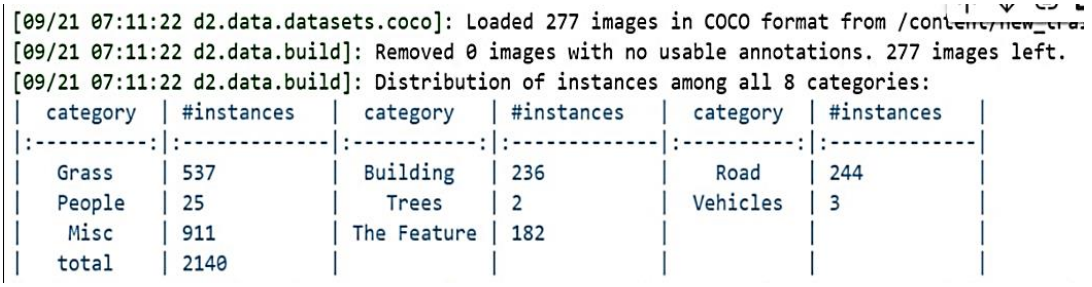
The loss function played a crucial role in measuring the difference between predicted and target values for objects. During the training process, there was a noticeable reduction in variance values from 3 to 1, indicating the algorithm's successful training.

When a loss function converges to a value of 1, it indicates that during the training process of a machine learning or deep learning model, the loss steadily decreases and approaches the specific numerical value of 1. In the context of model training, the loss function serves as a

critical metric for evaluating how closely the model's predictions align with the actual target values, often referred to as ground truth. A loss value of 1 typically implies a suboptimal level of accuracy, as it reflects a relatively high degree of error or disparity between the model's predictions and the true data. This convergence towards 1 suggests that the model may struggle to effectively capture the underlying patterns within the data, potentially necessitating adjustments to the model's architecture or training procedures to enhance its overall performance. It is essential to note that the interpretation of a loss value depends on the specific problem domain and dataset, with the primary goal being diligent monitoring of the loss function and other pertinent evaluation metrics to determine when the model achieves a satisfactory level of performance tailored to the given task.

To further improve the outcomes, I conducted multiple tests and added the results below for consideration:

A. In an effort to reduce the dataset size, we decreased the number of images from 927 to 463, effectively halving the dataset. Consequently, the model relied on only 277 images for training and 186 images for testing. Figure 6.5 visually represents the distribution of objects within this reduced training dataset.



[09/21 07:11:22 d2.data.datasets.coco]: Loaded 277 images in COCO format from /content/new_ds...
[09/21 07:11:22 d2.data.build]: Removed 0 images with no usable annotations. 277 images left.
[09/21 07:11:22 d2.data.build]: Distribution of instances among all 8 categories:

category	#instances	category	#instances	category	#instances
Grass	537	Building	236	Road	244
People	25	Trees	2	Vehicles	3
Misc	911	The Feature	182		
total	2140				

Figure 6.5. Indicate How Many Objects Which Have in this Training Case

The output was not acceptable as the loss function result was so low as Figure 6.6 shown.

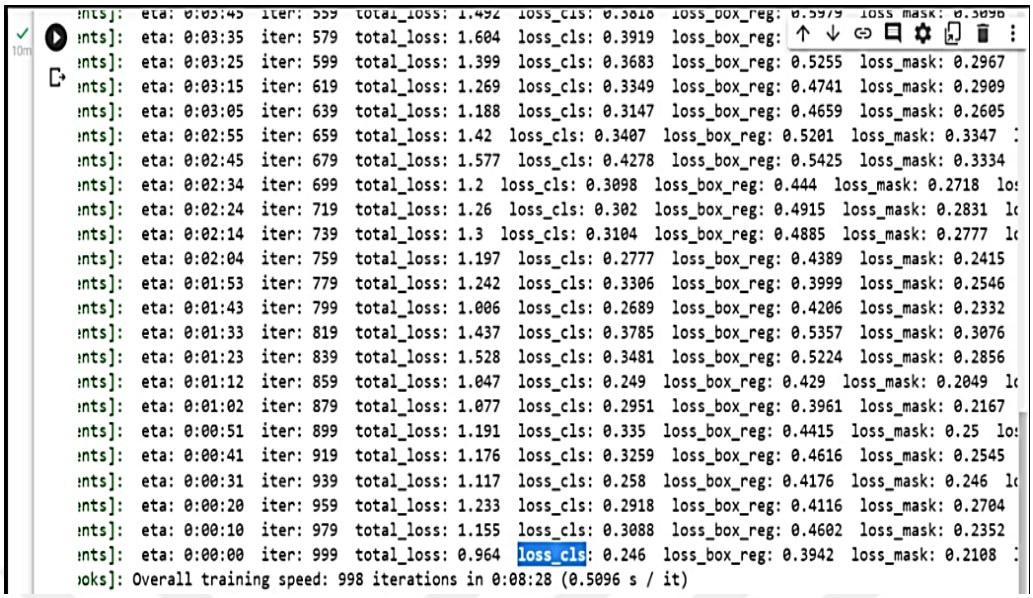


Figure 6.6. Loss Function Result

Training set result in classification and segmentation was around 91%.

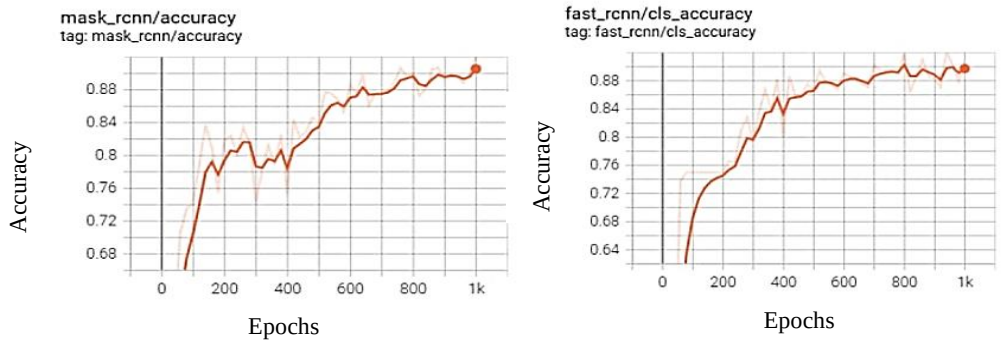


Figure 6.7. Training Set Result in Classification and Segmentation

Upon evaluating the test set, we observed a test result of approximately 50%. Interestingly, this case revealed that even after eliminating a portion of the dataset, the overall result did not show a significant deviation from the previous outcome, which was approximately 60% before the elimination.

B. Another approach we considered was working with only half of the data while also excluding the smallest object from the dataset. Among the objects present, the "Trees" object was notably small, prompting us to remove it. The resulting

dataset, depicted in Figure 6.8, contains objects excluding the "Trees" category. As observed, there are no "Trees" objects in this modified dataset.

```
WARNING [09/21 08:24:46 d2.evaluation.coco_evaluation]: COCO Evaluator instantiated using config, this is
WARNING [09/21 08:24:46 d2.data.datasets.coco]:
Category ids in annotations are not in [1, #categories]! We'll apply a mapping for you.

[09/21 08:24:46 d2.data.datasets.coco]: Loaded 186 images in COCO format from /content/new_test.json
[09/21 08:24:46 d2.data.build]: Distribution of instances among all 7 categories:
| category   | #instances   | category   | #instances   | category   | #instances   |
|:-----------|:-------------|:-----------|:-------------|:-----------|:-------------|
| Grass      | 340          | Building   | 169          | Road       | 172          |
| People     | 30           | Vehicles   | 4            | Misc       | 539          |
| The Feature | 113         |            |              |            |              |
| total      | 1367         |            |              |            |              |
```

Figure 6.8. Indicate How Many Objects Which Have in this Training Case

The output didn't change the result for loss function in classification and segmentation for the training dataset was almost the same as the last case, as Figure 6.9 shown

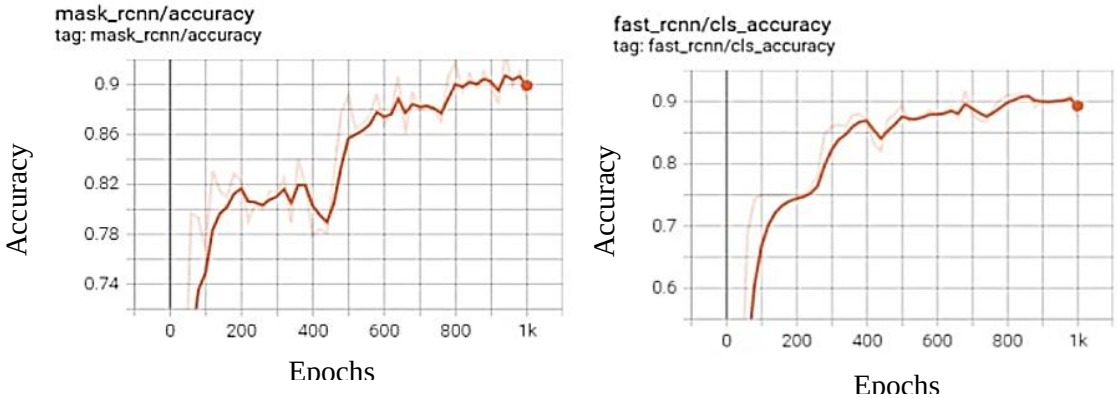


Figure 6.9. Training Set Result in Classification and Segmentation

So, as we noticed if we eliminate the object the number of classes will be less. That means the result will not be better than before (60%).

C. In another approach, we decided to eliminate all small objects, specifically "Trees," "People," and "Vehicles," from the entire dataset consisting of 927 images. Figure 6.10 illustrates the objects present in the training dataset after the removal of these smallest three categories. However, despite this modification,

the accuracy in the training set did not significantly improve and remained at the same level as in previous cases. This is depicted in Figure 6.11, where the training accuracy shows no notable increase from earlier results.

```
[09/22 15:08:18 d2.data.datasets.coco]: Loaded 186 images in COCO format from /content/new_test.json
[09/22 15:08:18 d2.data.build]: Distribution of instances among all 5 categories:
```

category	#instances	category	#instances	category	#instances
Grass	340	Building	169	Road	172
Misc	539	The Feature	113		
total	1333				

Figure 6.10. Indicate How Many Objects Which Have in This Training Case

So, the accuracy for test results didn't go up more than 55%. That means eliminating the smallest object or unbalanced dataset will not affect the result as we expected before.

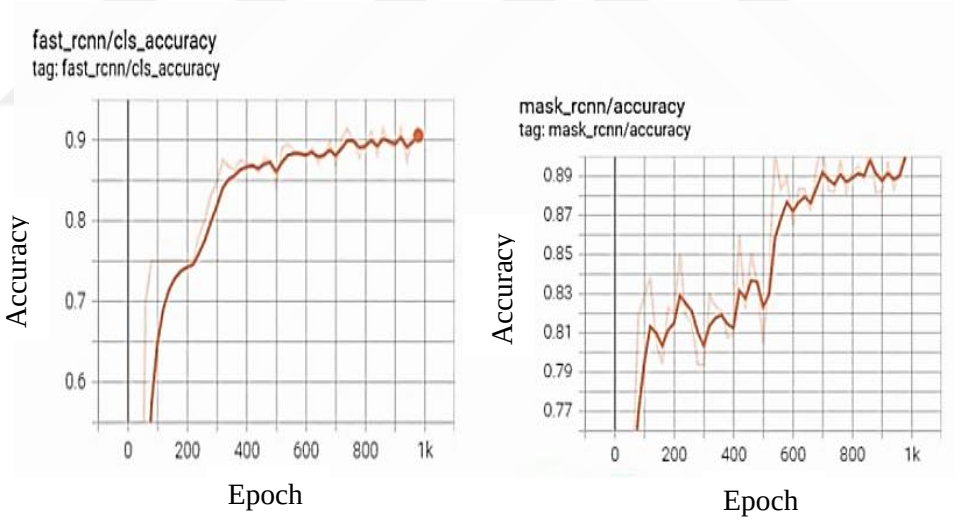


Figure 6.11. Training Set Result in Classification and Segmentation

6.2. U-Net architecture

On the contrary, we explored the implementation of the U-Net, a type of convolutional neural network (CNN). The U-Net architecture can be conceptualized as comprising an encoder network followed by a decoder network. In contrast to classification tasks, where the final output of the deep network is the primary focus, semantic segmentation demands both pixel-

level discrimination and a mechanism to project discriminative features learned at various encoder stages onto the pixel space.

One distinguishing feature of the U-Net is its loss function, which compares the predicted mask with the ground-truth mask. This enables parameter updates that lead to improved segmentation performance in subsequent training examples.

The fundamental concept behind the U-Net involves first obtaining a lower-dimensional representation of the image through a traditional convolutional neural network and then upsampling that representation to generate the final output segmentation map. Typically, U-Net models consist of 23 layers; however, the current model utilizes 19 layers.

Upon applying the U-Net model to a dataset comprising 927 images, we achieved an accuracy result of approximately 72%, outperforming Mask-RCNN in our segmentation task. Despite this higher accuracy, we are still eager to enhance the results further by applying the data augmentation mechanism. This ongoing pursuit aims to improve the model's robustness and overall performance in our segmentation endeavors.

6.2.1. Data Augmentation

Data augmentation is a crucial technique in the training of Convolutional Neural Networks (CNNs) that significantly improves model performance and generalization. By artificially enlarging the training dataset through various transformations, data augmentation exposes CNN to a more diverse range of examples, helping it become more robust and capable of handling unseen data. Data augmentation helps prevent overfitting and improves the generalization of CNN models by introducing variations in the training data. This allows the model to learn more robust features and patterns that are applicable to unseen data (Krizhevsky & Hinton, n.d.). It is seamlessly integrated into the CNN training process, where random samples of augmented data are fed to the model in each training iteration (Shorten & Khoshgoftaar, 2019).

Consequently, to address the challenge of a relatively small dataset size, we employed data augmentation techniques to effectively expand the training data. This approach involves creating new augmented samples by applying random shifts, rotations, variations in gray values, and random elastic deformations to the existing training data. As illustrated in Figure 6.12, the dataset size has been significantly increased to 3700 images, a substantial growth from the original 927 images. Data augmentation played a crucial role in enriching the training dataset and enhancing the model's ability to learn diverse patterns and improve overall performance.

```
Train Size: 2944 images
Test Size: 736 images
Val Size: 589 images
```

Figure 6.12. Shown the Data Set Has Increased To 3700 Images

The final accuracy rate is 82.8%. Comparing real images and its mask with the prediction value, we got a high near result as shown in Figure 6.13. And Figure 6.14 shows the accuracy chart.

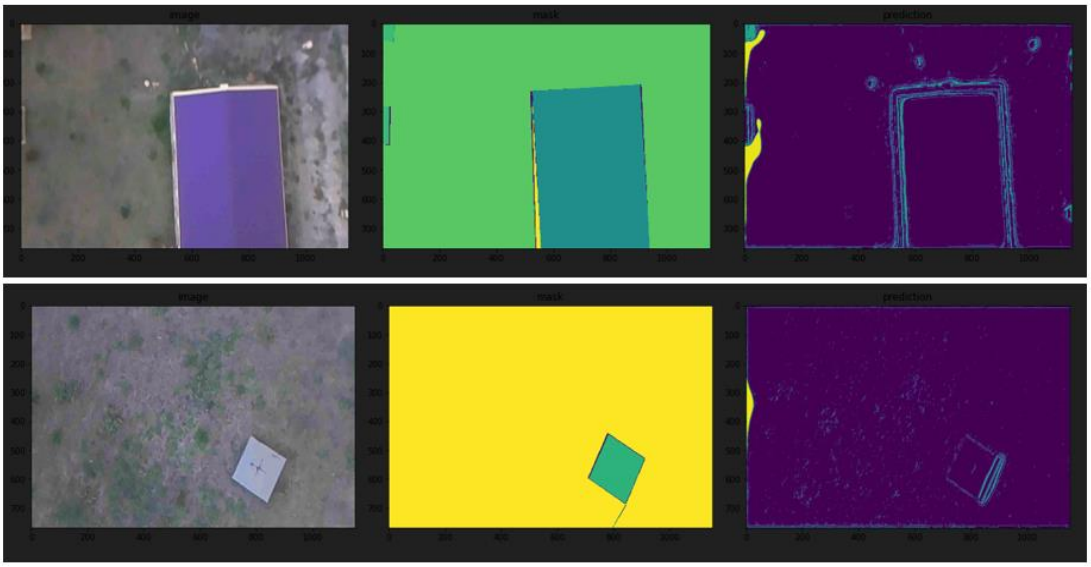


Figure 6.13. Real Images and its Mask with the Prediction Value

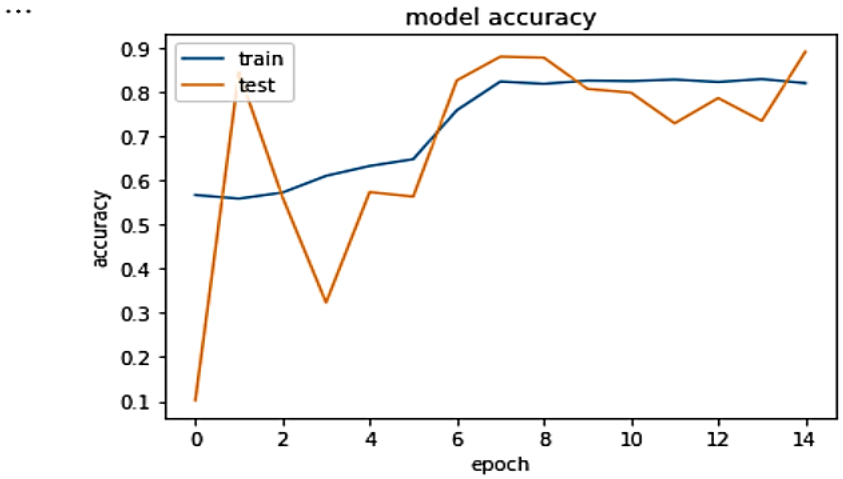


Figure 6.14. The Unet Accuracy Chart After Applied Data Augmentation

Therefore, and as a previous result in Figure 6.15, the best accuracy that got it, it was almost 82.8%.

```
588/588 [==============================] - ETA: 0s - loss: 0.0072 - accuracy: 0.825
Epoch 10: saving model to model.h5
588/588 [==============================] - 356s 605ms/step - loss: 0.0072 - accuracy: 0.8251 - val_loss: 0.00
Epoch 11/30
588/588 [==============================] - ETA: 0s - loss: 0.0077 - accuracy: 0.8240
Epoch 11: saving model to model.h5
588/588 [==============================] - 356s 606ms/step - loss: 0.0077 - accuracy: 0.8240 - val_loss: 0.00
Epoch 12/30
588/588 [==============================] - ETA: 0s - loss: 0.0075 - accuracy: 0.8275
Epoch 12: saving model to model.h5
588/588 [==============================] - 356s 605ms/step - loss: 0.0075 - accuracy: 0.8275 - val_loss: 0.00
Epoch 13/30
588/588 [==============================] - ETA: 0s - loss: 0.0074 - accuracy: 0.8219
Epoch 13: saving model to model.h5
588/588 [==============================] - 356s 606ms/step - loss: 0.0074 - accuracy: 0.8219 - val_loss: 0.00
Epoch 14/30
588/588 [==============================] - ETA: 0s - loss: 0.0073 - accuracy: 0.8286
Epoch 14: saving model to model.h5
588/588 [==============================] - 356s 606ms/step - loss: 0.0073 - accuracy: 0.8286 - val_loss: 0.00
Epoch 15/30
588/588 [==============================] - ETA: 0s - loss: 0.0076 - accuracy: 0.8194
Epoch 15: saving model to model.h5
588/588 [==============================] - 356s 605ms/step - loss: 0.0076 - accuracy: 0.8194 - val_loss: 0.00
Epoch 15: early stopping
<keras.callbacks.History at 0x7f78a61597d0>
```

Figure 6.15. The Accuracy Result for U-Net Data Augmentation

Subsequently, after augmenting the dataset to contain 3700 images, we observed significant improvements in the segmentation stages. Despite having an unbalanced dataset, wherein we

attempted various cases of eliminating small objects without achieving satisfactory accuracy, we realized that the issue of unbalanced objects was not the primary concern in our case.

Based on the results obtained, we can conclude that the U-Net architecture is the optimal choice for semantic segmentation tasks, while the Mask R-CNN architecture is better suited for instance segmentation. Therefore, we decided to adopt the U-Net architecture, merging it with the ResNet34 backbone, which led to the attainment of high-accuracy results.

The successful utilization of the U-Net architecture with the ResNet34 backbone highlights its effectiveness in our specific segmentation problem, and this combination has proven to be instrumental in achieving the desired level of accuracy and performance.

7. RESULTS AND DISCUSSIONS

As mentioned earlier in this thesis, the utilization of the patchify process resulted in doubling the dataset to 1852 photos. The data was then divided, allocating 20% for the test set and 80% for the training set. With these configurations, we proceeded with 30 training epochs and a learning rate of 0.00025. The outcome of these settings is illustrated in the results presented below.

7.1. Prediction

The semantic segmentation method discussed in this thesis generates pixel-wise predictions, providing probabilities for each class at every pixel. The model's performance is assessed by computing the total loss using Equation 3.2. After applying the patchify technique to the original images, which had dimensions of 1280 x 720 x 9, the model's output is now 512 x 512 x 9. In this output, the last dimension represents a vector that indicates the probability for each pixel corresponding to the class index. The first two dimensions of this vector represent the location of the pixel.

Consequently, the model effectively doubled the dataset, resulting in an increase in the number of RGB images from 927 to 1852, including both original and labeled photos. During the segmentation prediction phase, the output is produced at the same location as the zero tensors. The RGB values of the zero tensors in the dimensions are substituted. These zero tensors are relevant as each component is set to zero. To facilitate this process, the probability values need to be converted into RGB format, and the most significant index value is assigned to the class index. This algorithm then identifies the class index and transforms the corresponding category value into an RGB value.

Figure 7.1. Showcases some samples of the anticipated procedure taken from the model, providing a visual representation of the segmentation results.

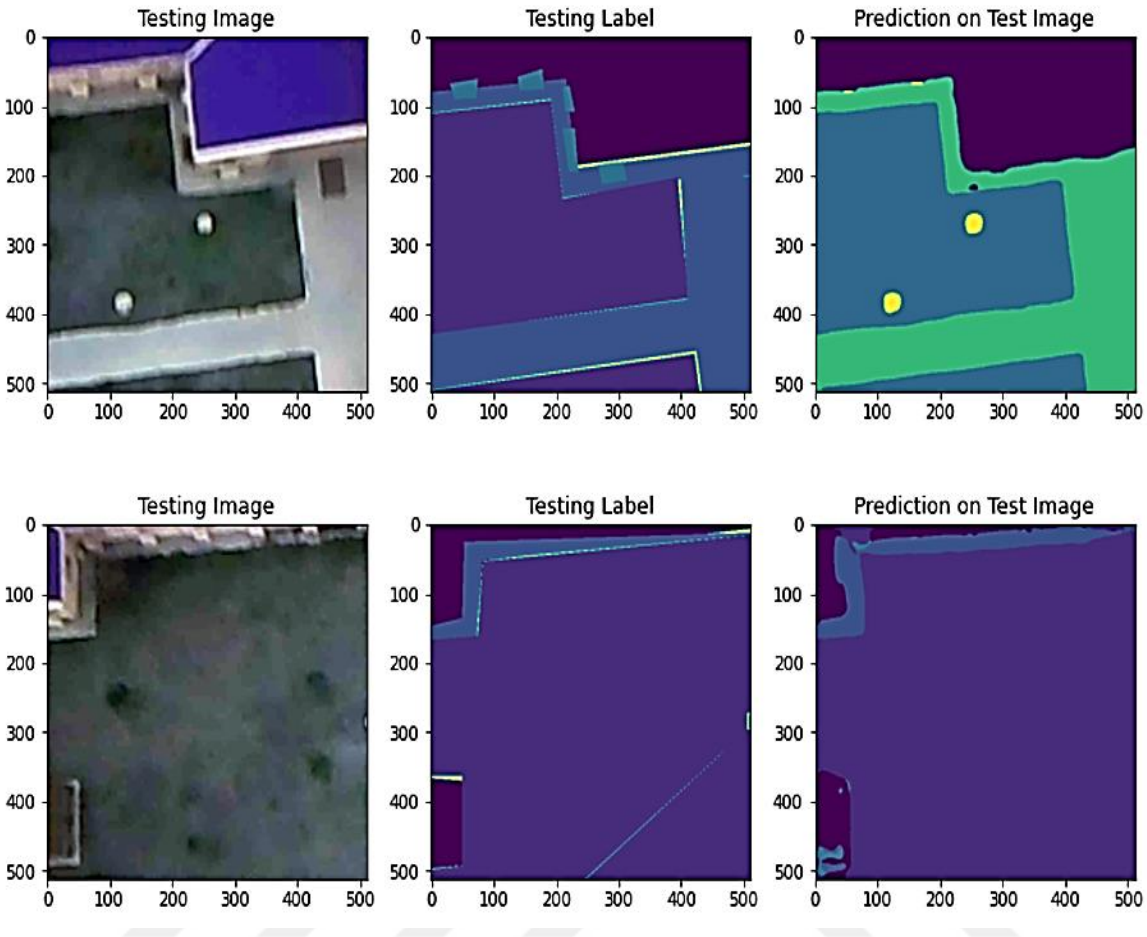


Figure 7.1. Examples of Some Output Images

7.2. Training result

In the training process, this model utilized 80% of the dataset, approximately 1480 images, to enable learning and predict the desired outcomes effectively. The training was conducted over 20 epochs, where each epoch represents a single forward and backward pass of the entire input data—a crucial step on which the algorithm relies in each training stage.

Since our model operates on the principles of supervised learning, it required labeled data before commencing the training. During the training phase, all classes were assigned equal weights to achieve more balanced results.

To build the architecture, we incorporated the ResNet34 network as a backbone, which had been previously trained on a diverse range of classes in large-scale image recognition tasks, mainly for the classification part. As shown in Figure 7.2, the training loss was recorded as 0.1034. Additionally, the training accuracy for this model was found to be 96.6%, while the validation accuracy was 96.34%, as depicted in Figure 7.3.

These results indicate that our model has achieved significant accuracy and demonstrates its capability in successfully predicting the desired outcomes during the training process.

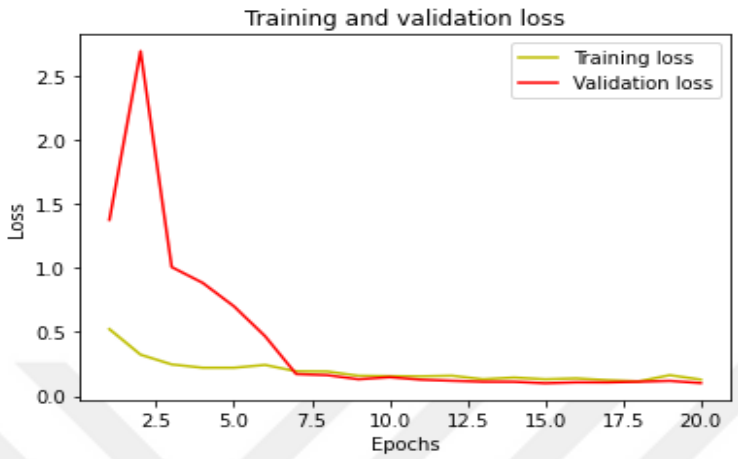


Figure 7.2. The Loss Function for Training Data

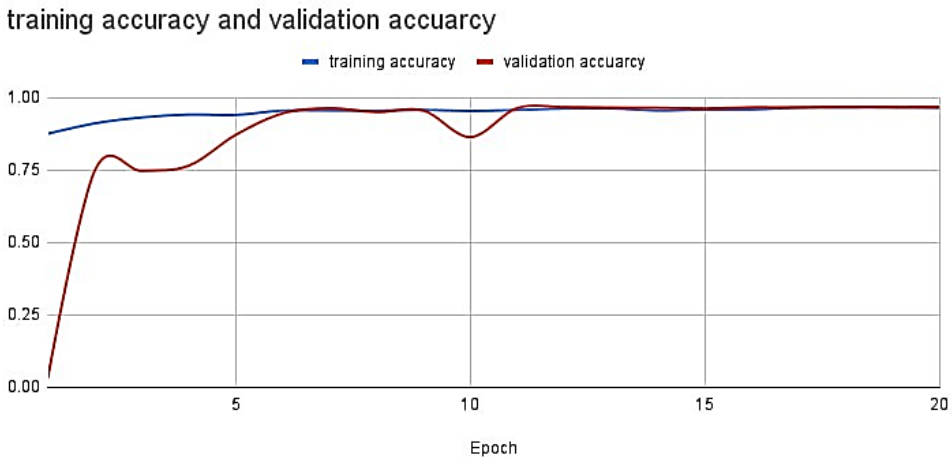


Figure 7.3. The Accuracy of Training and Validation

As depicted in Figure 7.2, the "Optimal Point" at 0.1034 denotes the precise location on the chart where the loss function achieves its lowest value. This particular point signifies the pinnacle of the model's performance when assessed against the validation data.

Regarding the observation of overfitting and underfitting, it is evident that during the initial 2.5 epochs, the graph exhibits characteristics of overfitting. Overfitting manifests when the loss on the validation data experiences an ascent. Subsequently, from approximately 2.5 epochs to

nearly 7.5 epochs, indications of underfitting emerge, leading to consistently elevated loss values.

A crucial aspect of the evaluation is the comparison of loss function values between the validation and training datasets. Notably, during the initial 7.5 epochs, a substantial disparity between the training and validation loss is discernible, signifying overfitting. However, as the algorithm progresses, a convergence occurs, resulting in a more equitable balance between the loss values computed on the validation and training datasets.

7.3. Test Result

For the testing stage, approximately 370 images, equivalent to 20% of the dataset, were used. This set of data serves as previously unseen data to evaluate the model's performance and efficiency. Assessing the model's accuracy on this test data is crucial in understanding how well the model can generalize to new, unseen data, which is essential for improving the machine learning model.

By referring to Figure 7.4, we can observe the accuracy of the model's predictions on the test data, which provides valuable insights into its performance. Additionally, Figure 7.5 illustrates the ratio of the loss function, indicating that it is relatively small. This small loss function ratio signifies a high success rate for the model's accuracy on the testing data.

These results demonstrate the effectiveness and efficiency of the model in predicting unseen data, reflecting its strong performance and capability to generalize to new instances.

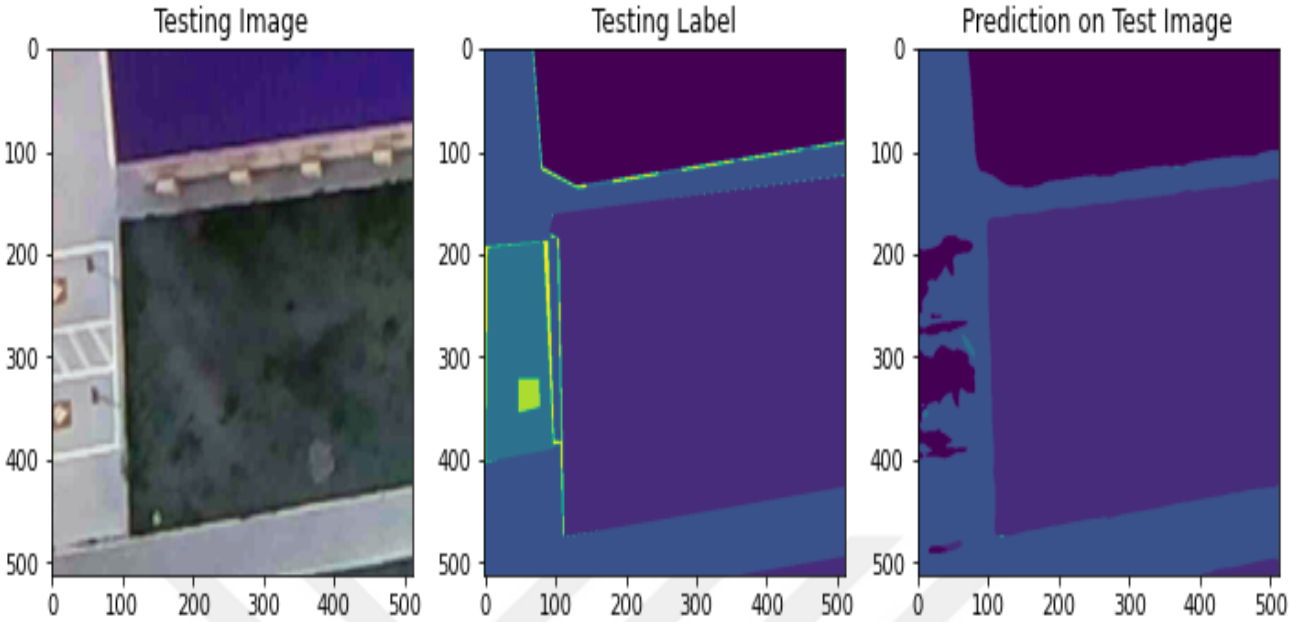


Figure 7.4. Example of Prediction Test Data

7.3.1. Result Of the Best Landing Site

The model successfully identifies the center pixels of each class, designating them as the optimal safe landing points, which are then enclosed within a red box. These boxes are added to the test images and showcased in the prediction results.

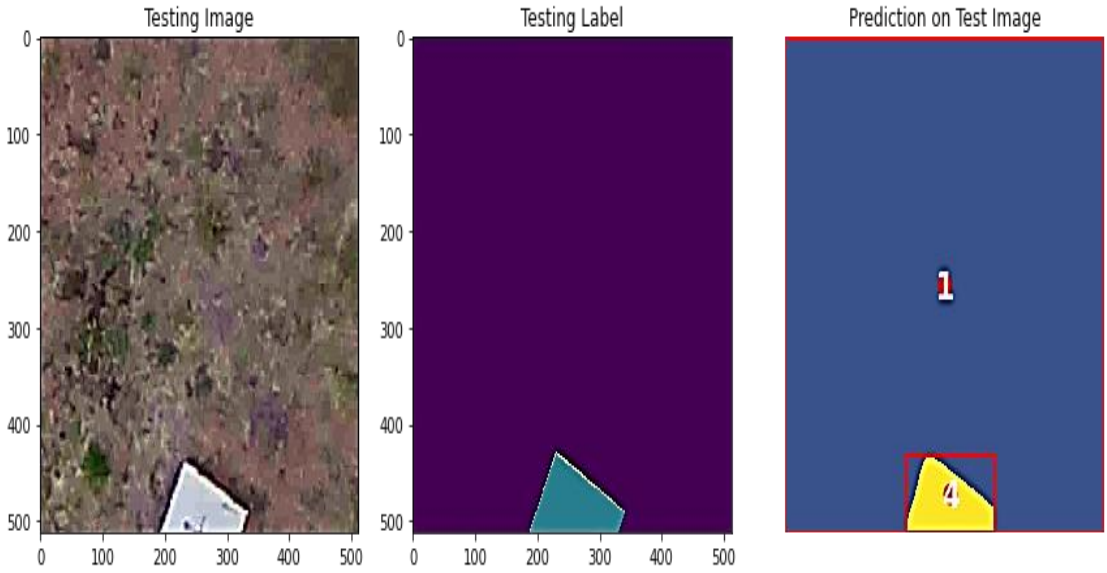


Figure 7.5. Surrounding Region with their Numbers

The center points are strategically chosen as the safest landing regions, situated away from the edges to minimize the risk of damage to the UAV during landing.

Figure 7.5 provides an illustrative example, where the model accurately determines the grass area as class number 1, demonstrating the capability of identifying suitable landing locations.

7.4. Result Of the Efficient Trajectory

The *A** algorithm determine the shortest route to reach the designated landing spot, as demonstrated in Figure 7.7. The model was trained on the available dataset, and the resulting predictions were evaluated on the test dataset.

Figure 7. 6. Path Planning to the Landing Site

Consequently, this model operates effectively in both path planning through image processing and ensuring safe landings in the event of engine failures, contributing significantly to drone safety. The foundation of our approach lies in path planning via image processing, serving as the primary mechanism for guiding the drone to its intended destination under typical operating conditions. However, the significance of this relationship increases in cases of engine failure.

Image processing-based path planning systems exhibit rapid adaptability, responding to the loss of engine function by orchestrating emergency landing strategies. These strategies account for critical factors, including altitude, available landing zones, and obstacle avoidance. In essence, path planning via image processing assumes a dual role: optimizing flight paths in routine scenarios and seamlessly transitioning to contingency plans, ensuring the secure descent of the drone when engine failure poses a threat to its ordinary operations. This collaborative synergy is paramount in elevating the safety and dependability of UAVs across real-world applications.

8. CONCLUSIONS

The thesis provided a method for processing images from a quadcopter camera to display location detection on a drone's vision system. The processing unit of Google Colab was used to train the convolution neural network model, which allowed the system to correctly identify a good landing spot and plan a landing trajectory. Verifying the outputs' outcomes and analyzing their analysis:

- The patchify feature was used to double the quantity of the dataset from 927 images to 1852 images since the more data we have, the more accurate the outcome would be.

- Convolutional neural network structures like Mask RCNN Fast RCNN U-Net were used in an attempt to classify and segment the images. Using this combination of the U-Net and ResNet Backbone 34 formula, we were able to classify and segment the images with accuracy during the training process estimated at 96.6%.

- There are a number of ways to choose a safe landing point, but for the dataset case of this thesis, we relied on picking the object's midpoint after enclosing it in a rectangle so that we could perform the necessary calculations to guarantee the distance from the edges, the safety of the UAV, and the safety of the surrounding environment.

- There were several options available to us when sketching the path, one of which was to specify multiple landing areas and make a trade-off, meaning that if the first object is discovered, it has precedence, if it is not found in these images, it will move to the other object, and so on. However, it was authorized to choose one of them.

- The A Star Algorithm is a rich algorithm, we also conclude. The choice of the heuristic function has a significant impact on the algorithm's effectiveness and efficiency. Therefore, in contrast to the DDA approach, the heuristic function provides a lower bound estimate of the target node cost without overestimating it. The most frequent inference is the Manhattan distance or Euclidean distance between the current node and the target node.

The quadcopter UAV was then given a safe landing trajectory that was both efficient and vision based.

9. RECOMMENDATIONS

Unmanned aerial vehicles (UAVs) must have the ability to independently locate and land at predetermined locations, and this capability is known as vision-based landing site detection. The following is a suggestion for a UAV's vision-based landing location detection system:

- Camera selection: The training process and the outcome will be more accurate if there is a high-resolution camera with an appropriate field of view (FOV) and image quality to record clear and detailed photos of the landing spot. Additionally, we need to think about things like weight, power usage, and compatibility with the UAV platform.

- Utilize pose estimation algorithms to determine the UAV's position and orientation in relation to the discovered landing place.

- Integration with Flight Control System: Creating an interface between the UAV's flight control system and the vision-based landing site detection system. Through this integration, the UAV is able to make judgments based on the vision system's real-time feedback and the discovered landing places. Make that the two systems are compatible and communicate effectively.

- Validation in the Real World: Examining the system's performance under actual operational circumstances. To confirm the performance and dependability of the vision-based landing site identification system, test it in various terrains, lighting situations, weather scenarios, and with various landing site characteristics.

- Continuous Improvement: Monitoring developments in UAV, machine learning, and computer vision. Update and develop the vision-based landing site identification system regularly to include fresh methods and formulas that boost precision, effectiveness, and security.

REFERENCES

Al-Najjar, H. A. H., Kalantar, B., Pradhan, B., Saeidi, V., Halin, A. A., Ueda, N., & Mansor, S. (2019). Land cover classification from fused DSM and UAV images using convolutional neural networks. *Remote Sensing*, *11*(12), 1–18.

Al-qurishi, M., Khalid, T., & Souissi, R. (2021). Deep Learning for Sign Language Recognition : Current Techniques , Benchmarks , and Open Issues. *IEEE Access*, *PP*, 1.

Ameli, Z., Aremanda, Y., Friess, W. A., & Landis, E. N. (2022). Impact of UAV Hardware Options on Bridge Inspection Mission Capabilities. *Drones*, *6*(3).

Aviation, F. (2016). *Remote Pilot – Small Unmanned Aircraft Systems*. *August*.

Ayhan, B., Kwan, C., Um, Y. Bin, Budavari, B., & Larkin, J. (2019). Semi-Automated Emergency Landing Site Selection Approach for UAVs. *IEEE Transactions on Aerospace and Electronic Systems*, *55*(4), 1892–1906.

Belge, E., Altan, A., & Hacıoğlu, R. (2021). Uyarlamali Bulanik Mantik Denetleyici Tabanli İnsansiz Hava Araci (İHA)'nin Rota Takibi Ve Faydali Yük Taşima Performansi. *Mühendislik Bilimleri ve Tasarım Dergisi*, *9*(1), 116–125.

Cesetti, A., Frontoni, E., Mancini, A., Zingaretti, P., & Longhi, S. (2010). A Vision-based guidance system for UAV navigation and safe landing using natural landmarks. *Journal of Intelligent and Robotic Systems: Theory and Applications*, *57*(1–4), 233–257.

Dag, T., Unluer, T., Copur, E. H., & Cakin, U. (2022). Kentsel Hava Taşimaciliğinda Kullanilacak Dikey İniş-Kalkiş Kabiliyetine Sahip Bir Hava Aracinin Kavramsal Tasarimi Ve Menzil Hesabi. *Konya Journal of Engineering Sciences*, *10*(3), 649–664.

Fakhraian, E., Semanjski, I., Semanjski, S., & Aghezzaf, E. (2023). *Towards Safe and Efficient Unmanned Aircraft System Operations : Literature Review of Digital Twins ' Applications and European Union Regulatory Compliance*.

Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., & De, D. (2019). Fundamental concepts of convolutional neural network. In *Intelligent Systems Reference Library* (Vol. 172, Issue January).

Giordan, D., Adams, M. S., Aicardi, I., Alicandro, M., Allasia, P., Baldo, M., De Berardinis, P., Dominici, D., Godone, D., Hobbs, P., Lechner, V., Niedzielski, T., Piras, M., Rotilio, M., Salvini, R., Segor, V., Sotier, B., & Troilo, F. (2020). The use of unmanned aerial vehicles (UAVs) for engineering geology applications. *Bulletin of Engineering Geology and the Environment*, *79*(7), 3437–3481.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transaction of Systems Science and Cybernetics*, *4*(2).

Huang, K. L., Chiu, C. C., Chiu, S. Y., Teng, Y. J., & Hao, S. S. (2015). Monocular vision system for fixed altitude flight of unmanned aerial vehicles. *Sensors (Switzerland)*, *15*(7),

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*(8), 5455–5516.

Krizhevsky, A., & Hinton, G. E. (n.d.). *ImageNet Classification with Deep Convolutional Neural Networks*. 1–9.

Mancini, F., Dubbini, M., Gattelli, M., Stecchi, F., Fabbri, S., & Gabbianelli, G. (2013). *Using Unmanned Aerial Vehicles (UAV) for High-Resolution Reconstruction of Topography: The Structure from Motion Approach on Coastal Environments*. 6880–6898.

Mat, U. A. V. L. (n.d.). *Technical Information Uav Landing Mat (Uavlm) Uav Landing Mat*.

Mualla, Y. (2021). *Explaining the Behavior of Remote Robots to Humans : An Agent-based Approach Yazan Mualla To cite this version : HAL Id : tel-03162833 Explaining the Behavior of Remote Robots to Humans : An Agent-based Approach*.

Mung, N. X., Nguyen, N. P., Pham, D. B., & Dao, N. N. (2022). *Synthesized Landing Strategy for Quadcopter to Land Precisely on a Vertically Moving Apron*. 1–14.

Pham, K. L., Leuchter, J., Bystricky, R., Andrle, M., Pham, N. N., & Pham, V. T. (2022). The Study of Electrical Energy Power Supply System for UAVs Based on the Energy Storage Technology. *Aerospace*, *9*(9).

Rayhan, A., & Kinzler, R. (2023). *Deep Residual Learning For Image And Video*. *August*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*.

Shah, A., Kadam, E., Shah, H., Shinde, S., & Shingade, S. (2016). Deep residual networks with exponential linear unit. *ACM International Conference Proceeding Series*, *21-24-Sept*, 59–65.

Shah Alam, M., & Oluoch, J. (2021). A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs). *Expert Systems with Applications*, *179*(March), 115091.

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1).

Telli, K., Kraa, O., Himeur, Y., Ouamane, A., & Boumehraz, M. (2023). *A Comprehensive*

Review of Recent Research Trends on. 1–48.

Thomas H. Cormen, Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. Massachusetts Institute of Technology.

Turgut, Z. (2020). *a Theoretical Comparison of Resnet and Densenet Architectures on the Subject of Shoreline Extraction*.

Yan, Y. (2023). Research on the A Star Algorithm for Finding Shortest Path. *Highlights in Science, Engineering and Technology*, *46*, 154–161.

Yeung, M., Sala, E., Schönlieb, C. B., & Rundo, L. (2022). Unified Focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, *95*(December 2021).

Yüzgeç, U., & Üçgün, H. (2021). Döner Kanatlı İHA'lar İçin Şarj Pedi Tasarımı ve Geliştirilmesi. *Global Conference on Engineering Research*, 318.

https://www.gatevidyalay.com/tag/a-algorithm-example-pdf/

https://www.dnv.com/to2030/technology/precision-agriculture.html

APPENDICES

Appendix A: ResNet Backbone 34 Python

```
import os
os.environ["SM_FRAMEWORK"] = "tf.keras"

from tensorflow import keras
import segmentation_models as sm

BACKBONE = 'resnet34'
preprocess_input = sm.get_preprocessing(BACKBONE)
# preprocess input
X_train_prepr = preprocess_input(X_train)
X_test_prepr = preprocess_input(X_test)

# define model
model_resnet_backbone = sm.Unet(BACKBONE, encoder_weights='imagenet',
classes=n_classes, activation='softmax')

# compile keras model with defined optimozer, loss and metrics
metrics=['accuracy', jacard_coef]
#model_resnet_backbone.compile(optimizer='adam', loss=focal_loss,
metrics=metrics)
model_resnet_backbone.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=metrics)

#print(model_resnet_backbone.summary())
callbacks = [
             keras.callbacks.ModelCheckpoint(
                os.path.join("./weights", "weights.03-unet-
resnet.{epoch:03d}.hdf5"),
                save_best_only=False,
                save_weights_only=True
              ),
            ]
# model_resnet_backbone.load_weights("./weights/weights.03-unet-
resnet.005.hdf5")
history2=model_resnet_backbone.fit(X_train_prepr,
          y_train,
          batch_size=8,
          epochs=30,
          verbose=1,
          validation_data=(X_test_prepr, y_test),
          callbacks=callbacks)
```

Appendix B: Determination of Landing site

```
# Predict on a few images

from skimage.measure import label, regionprops
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt
import random

test_img_number = random.randint(0, len(X_test))
test_img = X_test[test_img_number]
ground_truth=y_test_argmax[test_img_number]
# test_img_norm=test_img[:,:,0][:,:,None]
test_img_input=np.expand_dims(test_img, 0)
prediction = (model.predict(test_img_input))
predicted_img=np.argmax(prediction, axis=3)[0,:,:]

# Building 0
# Grass 1
# Road  2
# Misc 3
# The_Feature 4
# Trees 5
# Vehicles 6
# People 7
# background 8


fig, ax = plt.subplots(1, 3,figsize=(12, 8))
ax[0].imshow(test_img, interpolation = 'nearest')
ax[0].set_title('Testing Image')
ax[0].grid(False)

ax[1].imshow(ground_truth, interpolation = 'nearest')
ax[1].set_title('Testing Label')
ax[1].grid(False)

ax[2].imshow(predicted_img, interpolation = 'nearest')
ax[2].set_title('Prediction on Test Image')
ax[2].grid(False)

for region in regionprops(predicted_img):
    # take regions with large enough areas
    if region.area >= 100:

        print(region.label)
        # draw rectangle around segmented coins
        minr, minc, maxr, maxc = region.bbox
```

```
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='red', linewidth=2)
        h = minr - maxr
        w = maxc - minc
        center = (minc + (w /2), maxr + (h/2))
        circle = mpatches.Circle(center,radius=10, color='red')
        ax[2].add_patch(rect)
        ax[2].add_patch(circle)
        rx, ry = rect.get_xy()
        cx = rx + rect.get_width()/2.0
        cy = ry + rect.get_height()/2.0
        ax[2].annotate(region.label, (cx+1, cy+1), color='white',
weight='bold', fontsize=20, ha='center', va='center')


ax[2].set_axis_off()
plt.tight_layout()
plt.show()
```

Appendix C: Path Planning

```
import numpy as np
from skimage.measure import label, regionprops
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import random



# Get the center coordinates of the image
center_x = test_img.shape[0] // 2
center_y = test_img.shape[1] // 2

# Find the target position labeled as 'The_Feature'
label_img = label(predicted_img)
regions = regionprops(predicted_img)
target_region = None
for region in regions:
    if region.label == 4:  # 'The_Feature'
        target_region = region
        break
try:
  if target_region is None:
      print("Target position 'The_Feature' not found in the image.")

  # Plotting
  fig, ax = plt.subplots(1, 1, figsize=(8, 8))
  ax.imshow(test_img, cmap='viridis')
  ax.set_title('Testing Image')

  # Plotting the ground truth label
  ax.imshow(ground_truth, alpha=0.5, cmap='Set1')

  # Plot a straight line from the center to the target position if available
  if target_region is not None:
     center = ( center_x,center_y)
      target = (int(target_region.centroid[0]), int(target_region.centroid[1]))

      line = plt.Line2D([center[1], target[1]], [center[0], target[0]], color='red', linewidth=2)

      ax.add_line(line)

  else:
      print("Target position 'The_Feature' not found in the image.")

  # Add rectangle and circle for target region
  rect = mpatches.Rectangle((target_region.bbox[1], target_region.bbox[0]),
                            target_region.bbox[3] - target_region.bbox[1],
```

```
                                target_region.bbox[2] - target_region.bbox[0],
                                fill=False, edgecolor='red', linewidth=2)
    ax.add_patch(rect)
    circle = mpatches.Circle((target_region.centroid[1],
target_region.centroid[0]), radius=10, color='red')
    ax.add_patch(circle)

    ax.set_axis_off()
    plt.tight_layout()
    plt.show()
except for Exception as e:
    print(e.args)
```