Segmentation of Aerial Images for Drone Landing Detection

Davide Vettore
Artificial Intelligence for Science and Technology
University Milano Bicocca
Milan, Italy
d.vettore@campus.unimib.it

***Abstract*— This project presents a method for drone landing detection using semantic segmentation of aerial images. We train deep learning models, including UNet and PSPNet, on the Semantic Drone Dataset to accurately identify various terrain types. Our fine-tuned UNet achieves the highest performance, effectively identifying safe landing zones while avoiding obstacles.**

***Index terms*—Semantic Segmentation, Semantic Drone Dataset, UNet, PSPNet, mIoU**

I. Introduction

In recent years, drones have become increasingly integrated into various scenarios, from surveillance to package delivery. However, ensuring their safe and autonomous operation in urban environments remains a significant challenge. One critical aspect is accurately understanding the surroundings, which involves identifying different objects and terrain types visible from an aerial perspective.

A. *The dataset*

The Semantic Drone Dataset [1] addresses the challenge by providing a collection of 400 high-resolution images captured from altitudes between 5 to 30 meters above ground. Each image is annotated with segmentation masks, detailing 24 distinct classes such as paved areas, vegetation, vehicles, and obstacles.

B. *The task*

The experiment focuses on training deep learning networks for semantic segmentation using the Semantic Drone Dataset. Semantic segmentation involves assigning specific labels to each pixel in an image, allowing precise identification of objects and terrain features in the scene. Our goal is to develop and assess various neural network architectures like UNet and PSPNet. These models will be trained on a subset of 300 annotated images, with validation and testing performed on 50 images each, aiming to achieve highly accurate segmentation.

In addition to segmentation, another aspect of this project involves developing an algorithm to autonomously identify safe landing spots for drones. This algorithm will operate on the automatically segmented images to identify suitable landing zones away from potential obstacles such as buildings and trees.

Through this project, we aim to contribute to advancing autonomous drone technologies by demonstrating effective techniques for scene understanding and safe landing zone detection in urban environments.

II. Data Preparation

The Semantic Drone Dataset includes a variety of scenarios, ranging from complex urban environments captured at different altitudes to more rural, grassy areas. The dataset's annotations are notably accurate, capturing diverse elements such as buildings, vegetation, and other key features across these varied scenes.

Two sample images and the relative ground truth masks are shown in Figure 1.

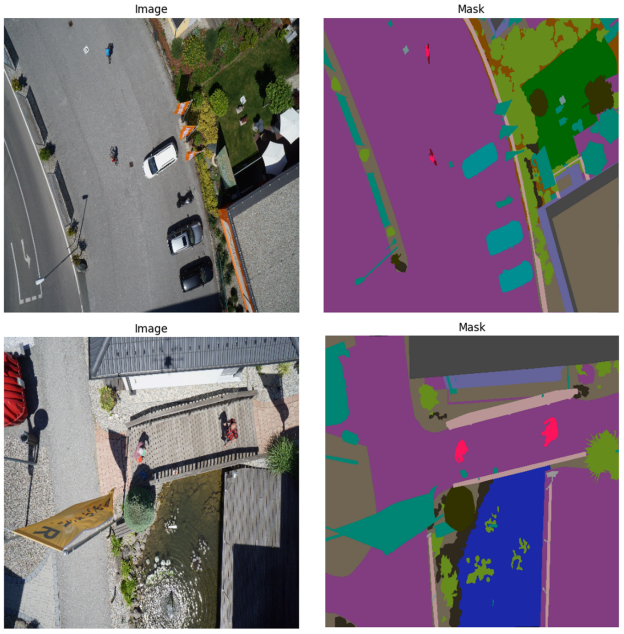

Figure 1: Sample images from the training set with their corresponding ground truth masks.

The two images both cover urban scenarios and demonstrate the dataset's range in altitude, likely representing the higher and lower limits. The ground truth segmentations is extremely sharp and detailed, reflecting the precision of the dataset.

The first image features a large paved area, which is the most common class overall, as well as vehicles like bicycles and cars. It also includes a residential area with a private garden, highlighting the variety of elements captured in urban settings.

The second image is more complex and includes a water feature with multiple types of vegetation as well as a large flag, which is segmented as an obstacle. The class distribution of the second image can be seen in Figure 2.

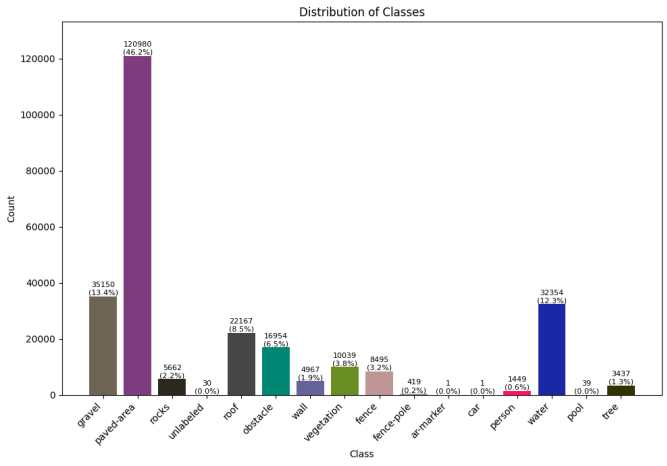

Figure 2: Class distribution of the second image in Figure 1.

After visualizing the variety of scenarios in the dataset, the next step involves loading the images and masks, along with their corresponding class labels and RGB values. The data is then split into training, validation, and test sets to ensure robust model evaluation. A custom dataset class is created, and data loaders are used for efficient batch processing during model training and evaluation. To manage computational resources effectively, the images are processed in batches of 16. The images and masks are then resized to 256x256 pixels, converted to RGB format, and normalized, ensuring consistency and efficient processing during training.

During the data preparation, we also defined an augmentation procedure that is applied when loading the training set. Data augmentation is a technique commonly used in training machine learning models, especially for image-based tasks like semantic segmentation. The main purpose of data augmentation is to artificially increase the size of the training dataset by creating modified versions of the original images. This helps the model generalize better to new, unseen data by introducing variability and reducing overfitting. The performance of the models will be evaluated both with and without augmentation to assess the impact of this technique on the training process.

In our project, we employ the Albumentations library for data augmentation [2]. The specific augmentations used include:

- Horizontal Flip: Randomly flips the image horizontally with a probability of 0.5. This helps the model learn that objects can appear in different horizontal orientations.
- Vertical Flip: Randomly flips the image vertically with a probability of 0.5, adding another layer of variability.
- Random Rotate 90: Rotates the image by 90 degrees randomly with a probability of 0.5. This ensures the model can handle different rotations of objects.

These transformations are only applied to the training data to make the segmentation model more robust and improve its performance. By focusing on spatial-level transformations, we avoid complications with pixel-level transformations on the masks, keeping the segmentation labels correct while still making the training images more diverse. In Figure 3 an example of the possible spatial-level transformations is shown.

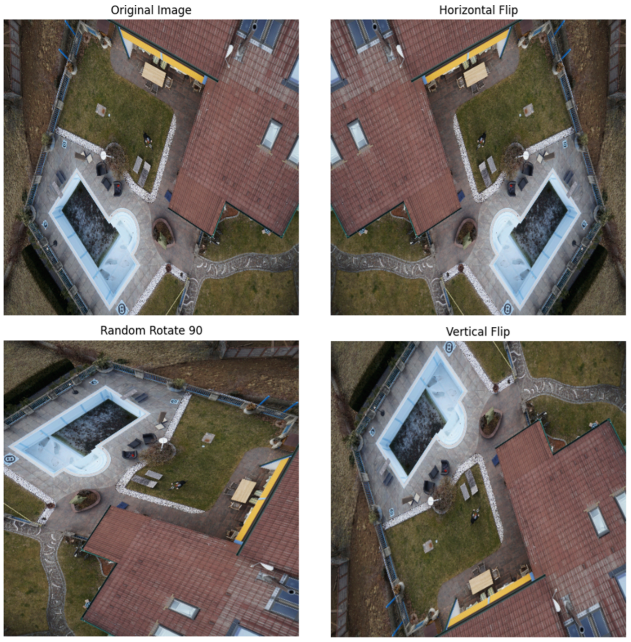

Figure 3: Example of spatial-level transformations applied on the training set.

III. Semantic Segmentation Methodologies

For the semantic segmentation task, we chose the UNet [3] architecture due to its well-documented effectiveness in pixel-level classification and its versatile application across various fields. UNet, introduced in 2015, was initially designed for biomedical image segmentation but has since found applications in many areas, including remote sensing, autonomous driving, and more.

We will then explore fine-tuned versions of the classical UNet and PSPNet [4] for comparison. Both of these pretrained models represent state-of-the-art approaches in semantic segmentation and will be implemented using the Segmentation Models PyTorch library [5].

A. *UNet*

A UNet is a fully convolutional network whose encoder-decoder structure, coupled with skip connections, allows to capture both low-level and high-level features, which is essential for precise segmentation. The encoder part reduces the spatial dimensions and extracts features, while the decoder part upsamples these features to reconstruct the segmentation map, ensuring detailed and accurate segmentation. A UNet architecture is presented in Figure 4.

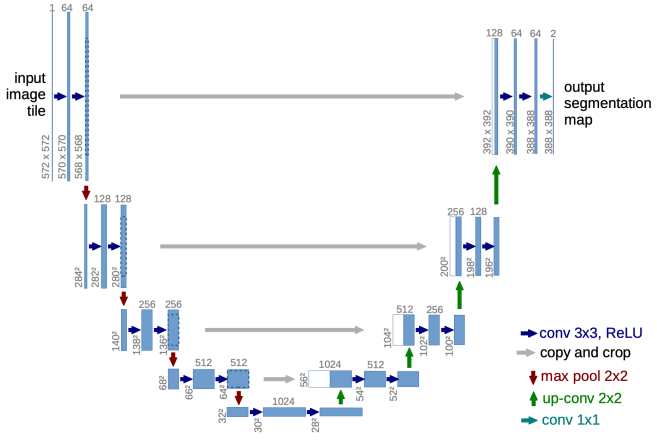

Figure 4: Typical UNet architecture presented in [3].

To implement our UNet model, we designed an architecture that effectively handles our images and masks. This architecture consists of several key building blocks that work together to achieve precise segmentation:

- *DoubleConvBlock:* This block is used throughout the network to perform convolution operations. Each block contains two convolutional layers, each followed by batch normalization and a ReLU activation function. This design allows the model to learn more complex features.
- *DownsampleBlock:* Found in the encoder part, each downsample block consists of a DoubleConvBlock followed by a max-pooling layer to reduce spatial dimensions by half, and a dropout layer to prevent overfitting. These blocks progressively reduce the spatial size of the feature maps while increasing the number of channels.
- *UpsampleBlock:* Located in the decoder part, each upsample block uses a transposed convolution layer to increase the spatial dimensions. It then concatenates the upsampled features with corresponding feature maps from the encoder (skip connections) before applying a DoubleConvBlock. A dropout layer is also included to reduce overfitting.
- *Bottleneck:* Positioned at the lowest level of the U-shaped architecture, it is a DoubleConvBlock that processes the most compressed representation of the input image.
- *Final Convolution Layer:* The last layer of the network is a single convolutional layer that maps the high-dimensional features to the desired number of classes (24 in this case).

The architecture built with these blocks has 34,526,616 parameters and is designed to handle 256x256 input images, ensuring precise and effective segmentation. The training will be performed using the Adam optimizer with a learning rate of 0.0001 and CrossEntropyLoss as the criterion, both implemented using PyTorch [6].

B. *Fine-tuned UNet*

For the fine-tuned UNet, we use a pretrained model with a ResNet-50 backbone, which acts as the encoder, with weights initialized from the ImageNet dataset [7]. While transfer learning usually involves keeping the encoder frozen with its pretrained weights, we instead use these weights as a starting point and retrain the encoder as well. This approach does make the training process slower, but it allows the model to learn more task-specific information. The UNet is customized for our specific task by setting the number of output classes to match our dataset, this model has a total of 32,524,440 trainable parameters.

The Adam optimizer, with a learning rate of 0.0001, is used for training, while the Dice Loss function is employed to handle the multiclass segmentation. The Dice Loss, based on the Dice-Sørensen coefficient, measures the overlap be-

tween predictions P and ground truth G, ensuring accurate segmentation across all classes:

$$\text{Dice}(P, G) = \frac{2\sum_i p_i g_i}{\sum_i p_i^2 + \sum_i g_i^2}$$

Both the model and the loss function are implemented using the segmentation models PyTorch library [5].

C. *Fine-tuned PSPNet*

We chose to utilize the Pyramid Scene Parsing Network for our segmentation task due to its ability to effectively capture both local and global context information, which is crucial for accurate scene parsing[1].

The PSPNet achieves this through its pyramid pooling module, which aggregates contextual information from different regions of the image. This approach allows the model to handle diverse and complex scenes, making it well-suited for our task. The PSPNet has demonstrated state-of-the-art performance on several benchmarks, including the PASCAL VOC 2012 [8] and Cityscapes datasets [9], proving its robustness in challenging environments. The architecture proposed by H. Zhao et al. is illustrated in Figure 5.

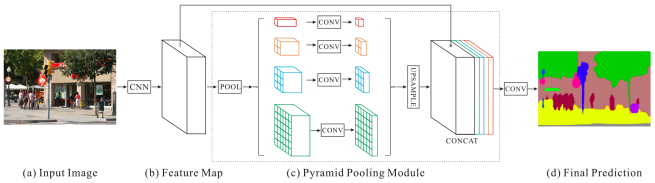

Figure 5: PSPNet architecture proposed in [4].

For this model, we also use a pretrained version with a ResNet-50 backbone initialized with ImageNet weights. The same optimizer and loss function described for the fine-tuned UNet is applied here as well. The introduced model has a total of 24,407,000 trainable parameters, making it the smallest among the models considered.

IV. Results

The training of all models was performed over 100 epochs, during this process, the train and validation loss and accuracy were monitored to track the models' performance and convergence. The training metrics of the three models are presented in Figure 6, we can see a clear convergence in all of them. The manually defined UNet and PSPNet models converge faster, showing a steeper drop in training loss. The fine-tuned UNet, however, shows a slower and more gradual decrease in loss and a slower increase in accuracy, suggesting that additional epochs might have further improved its performance compared to the other models.

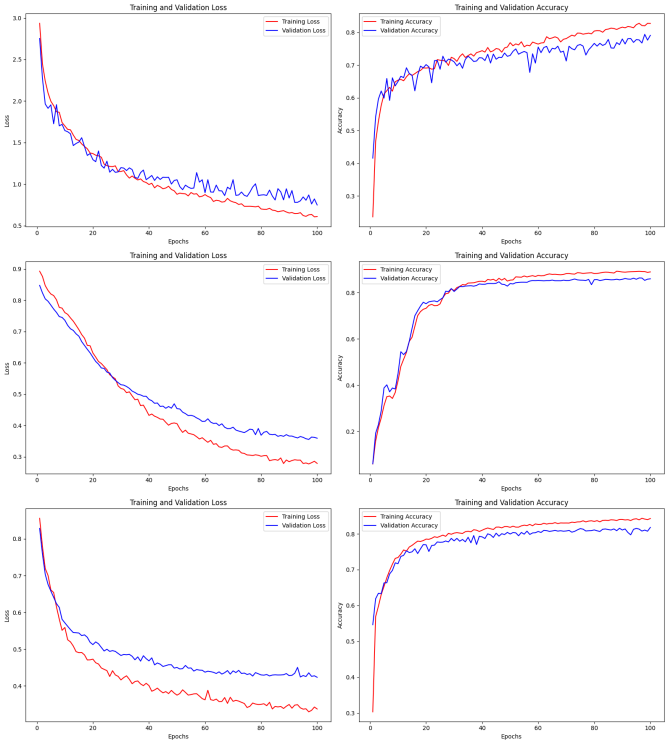

Figure 6: Comparison of training loss and accuracy across three models: UNet, fine-tuned UNet, and PSPNet.

The models were evaluated using several metrics: mean Intersection over Union, Dice coefficient, and pixel-wise accuracy. These metrics provide a comprehensive view of the models' ability to accurately segment the images. The IoU, also known as Jaccard similarity coefficient, measures the overlap between predictions P and ground truth G:

$$\text{Jaccard}(P, G) = \frac{|P \cap G|}{|P \cup G|} = \frac{|P \cap G|}{|P| + |G| - |P \cap G|}$$

The mean IoU is calculated as the average of all the class-wise IoUs. The Dice coefficient, already introduced as the loss function, evaluates the similarity between the predicted and actual segmentations. Pixel accuracy, on the other hand, reflects the percentage of pixels in the segmented image that are correctly classified, offering a general measure of prediction accuracy.

Table 1 presents the mean IoU scores achieved by the three architectures on the test set. As mentioned earlier, the models were trained both with and without spatial-level augmentation to evaluate the impact of this technique.

[1]Semantic segmentation and scene parsing both refer to labeling each pixel with a specific category. In our context, they are equivalent.

mIoU	UNet	Fine-tuned UNet	Fine-tuned PSPNet
Augmentation	41.78%	**58.01%**	46.88%
No Augmentation	41.52%	53.00%	45.51%

Table 1: Comparison of the testing mIoU for the three models trained with and without data augmentation.

By examining the mean IoU, it is clear that the best-performing model is the Fine-tuned UNet, which significantly outperforms the others. The data augmentation strategy improves the performance of all models, although most improvements are modest.

In Figure 8 we can now explore the class-specific metrics of our best-performing model to better understand its strengths and the challenges.

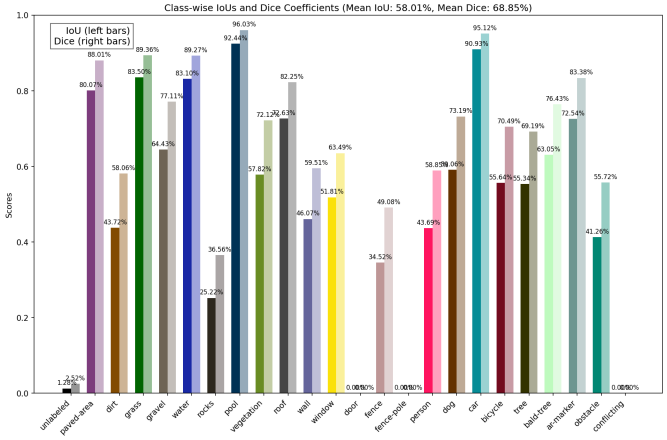

Figure 7: Class-specific IoUs and Dice coefficients achieved with the Fine-tuned UNet on the test set.

We observe both highs and lows in the results. The class with the highest IoU and Dice coefficient, indicating the best segmentation by our model, is *pool*, achieving an impressive 92.44% IoU, along with a 96% Dice coefficient and 96% accuracy. *Water* and various terrains, such as *paved-areas*, *grass*, and *gravel*, also perform well, while *dirt* has the lowest score among them. The vegetation features show impressive results as well; the metrics for *vegetation*, *trees*, and *bald trees* indicate that the model has learned to distinguish these similar features. Vehicles generally receive high scores, but the performance on urban features varies. The model excels at segmenting *roofs* and *windows* but struggles with *walls* and *fences*. It appears to completely neglect *doors* and *fence-poles*, suggesting that the model was unable to learn these structures during training.

Important classes such as *people* and *dogs* yield underwhelming results; we may need to refine our approach when working with living creatures, although this largely depends on the specific task. The model has difficulty segmenting *rocks* and also struggles with the more general *obstacle* class. The two additional classes, *unlabeled* and *conflicting*, achieve poor results, likely because they represent a minority in both the training and test sets. Overall, the results are promising, and we can now examine some examples for a more observational evaluation.

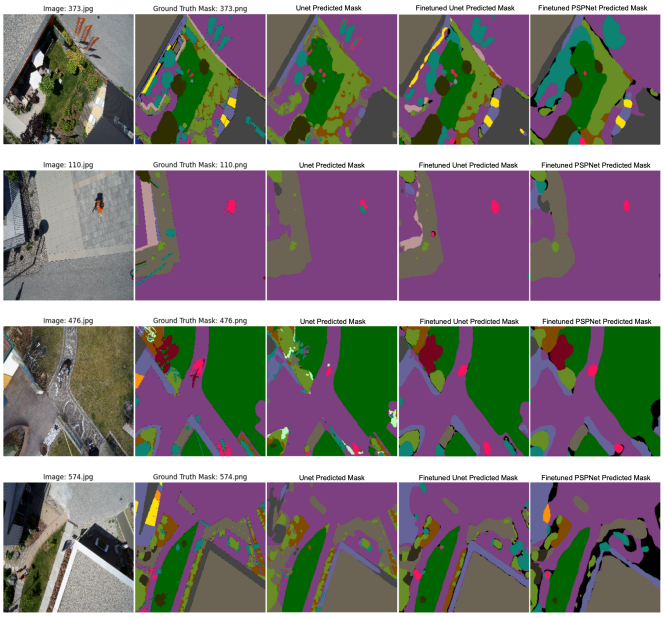

Figure 8: Comparison of ground truths and model segmentations of four sample images from the test set.

In the first image, which appears to be the most complex, almost all urban features are present, along with a lot of mixed vegetation. As anticipated by the previously observed metrics, the fine-tuned UNet delivers the best performance. It's the only model that accurately segments all the windows, a particularly challenging class. Additionally, it correctly identifies three out of the four people in the image, with overall segmentation that is notably more accurate. The manually defined UNet, while making some sharp segmentations, struggles to distinguish between obstacles and people, especially with the bright orange flag. Interestingly, the PSPNet produces smoother segmentations, which aligns with its scene parsing approach. This method focuses more on dividing the entire scene into segments and assigning them class labels, rather than classifying each pixel individually. While this results is less detailed, it still performs adequately by identifying most of the windows and people in the image.

In the second image, which features numerous fences, a particularly challenging class, the fine-tuned UNet is the

only model that successfully identifies them, even with some difficulty in achieving precise segmentation. Even the best-performing model has trouble with shadows, mistakenly identifying them as a wall in the top left and a person in the middle. The PSPNet exhibits similar issues. However, all models handle the separation between paved areas, gravel, and vegetation well.

The third image is particularly interesting, as none of the models detect the bicycles being ridden, and only the fine-tuned models find the parked ones. The manually defined UNet seems more prone to identifying bald trees, but only one of its detections is correct. Once again, the PSPNet generates smoother segmentations.

In the last image, which contains smaller details due to being captured from a higher altitude, the various terrains present significant challenges for the PSPNet. It segments a large portion of the scene as unlabeled, a class barely used by the other models. However, PSPNet is the only model that nearly identifies a door, one of the more difficult classes overall. In this particular example, the manually defined UNet seems to offer the most visually accurate segmentation with respect to the ground truth.

In summary, most of the challenging classes pose difficulties for all the models. The fine-tuned UNet consistently gives the most accurate segmentations, especially for tough classes like windows and people. The PSPNet's scene parsing approach results in smoother outputs, which can be helpful in some cases, but it may miss some details. The manually defined UNet, while struggling with certain classes and having lower metrics, sometimes provides more visually consistent segmentations.

With these results in mind, we can now move on to the next task, where we will focus on finding the best landing spot for the drones. We'll use the Fine-tuned UNet as it is the best performing model overall, both in terms of metrics and observational evaluation.

V. Drone Landing Detection

In addition to segmentation, another aspect of this project involves developing an algorithm to autonomously identify safe landing spots for drones. This algorithm operates on the automatically segmented images to identify suitable landing zones away from potential obstacles such as buildings and trees. To achieve this, we manually define which of the 24 classes are safe for landing and which must be avoided. Safe classes include areas like paved and grassy areas, while avoid classes include potential obstacles such as roofs, trees, and sensitive classes like people.

The algorithm creates masks for both the avoid classes and safe classes. These masks are used to generate a distance transform map, computed with the SciPy Euclidean distance transform [10], which calculates the distance from each pixel to the nearest avoid class. This helps in identifying areas that are far enough from obstacles to be considered safe for landing. Additionally, a distance transform is performed from the borders of the image under the hypothesis that we don't know what is near the border, possibly some avoid areas. This ensures that the landing spot is not too close to the edges, which could be risky for the drone.

The two distance transforms are then combined to create a composite distance map. This map highlights the safest areas for landing by showing regions that are both far from obstacles and not too close to the borders.

The algorithm then scans through the image, checking for potential landing spots of a defined size. It evaluates each spot by calculating the average safety score from the combined distance map. The spot with the highest score is selected as the best landing position.

In the end, the algorithm returns the best landing spot coordinates, the safe mask, and the combined distance transform heat map. Figure 9 shows an example demonstrating the output of this algorithm applied to three sample images from the test set.

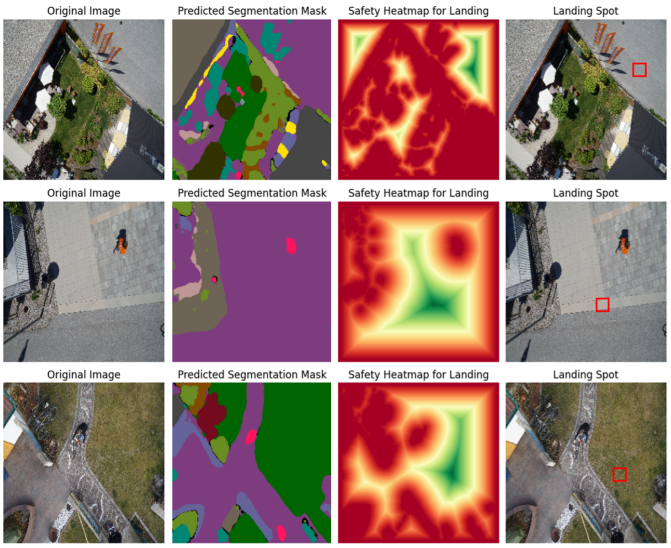

Figure 9: Example of the output produced by the algorithm on three sample images from the test set.

In the first image, the segmentation appears well done. The model successfully captures most of the building features like walls, windows, and roofs, which are sometimes difficult to detect. It also effectively distinguishes between trees, grass, dirt, and other types of vegetation, such as

bushes. The safety heatmap appears accurate from an observational standpoint. Notably, given the shadow, it seems that the gravel area in the top left might actually be part of a roof, although the model does not pick up on this detail. Despite this, the landing spot found by the algorithm seems accurate, positioned on the road away from trees and obstacles.

In the second image, there is a large area of pavement, where the model excels, correctly segmenting gravel and vegetation features. However, it struggles with the fence and mistakes a shadow for a wall. As seen in the class-wise IoUs, these are some of the more challenging classes to detect. The algorithm has no trouble identifying the person in bright orange and finds a landing spot that is undoubtedly safe.

The third image also includes some complex objects. The model segments the typical terrain features like gravel, grass, and dirt perfectly, but still confuses some fences as walls. Interestingly, the parked bicycles are correctly segmented, but the bicycles being ridden by people are not detected. Even in this example, the heatmap looks accurate, and the drone is safely directed to land on a grassy area.

Even though the segmentations are not perfect, the Fine-tuned UNet reliably handles most of the terrain classes and accurately identifies the majority of the avoid classes. As a result, the algorithm finds safe landing spots in all the test images. A notable limitation is that, when roofs are covered in gravel, the model classify them as safe landing zones. While this could complicate the drone's recovery, the landing would still be safe overall.

VI. Manually obtained aerial images

To test the generalizing abilities of the model and see if it's useful in real-world situations, we decided to manually process aerial images and observe the model's performance. The images were taken with a DJI Mini 4 Pro drone in a rural area of northern Italy. They were captured at heights ranging from 5 to 30 meters to match the dataset's conditions as well as comply with Italy's drone legislation. However, for the sake of generalization, we didn't check if the model's trained classes cover all the images.

Since there is no ground truth available for our evaluation, we will rely on an observational analysis to assess the performance of our model on newly obtained images. We collected a set of 20 aerial images, the more interesting results from our observational analysis are illustrated in Figure 10.

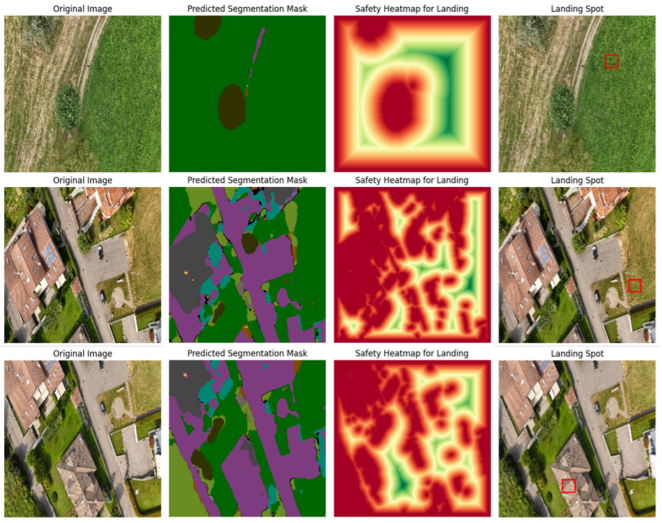

Figure 10: Three notable examples of segmentations and identified optimal landing spots from manually processed aerial images, evaluated without ground truth.

The first image demonstrates the model's ability to clearly segment grass, trees, and what appears to be a paved or dirt area, successfully identifying an optimal landing spot in the grass, away from any obstacles. The model performs similarly well in most aerial images with high vegetation density, accurately identifying safe landing zones.

In the second image, the segmentation is less sharp and precise. However, the model effectively distinguishes between paved areas, grass, obstacles, and roofs. An interesting observation is that the model classifies some skylights, which were not present in the training set, as windows, the most similar class available. Despite these challenges, the model still finds an optimal landing spot on the grassy area, far from cars and roofs.

The third example highlights the most notable error in the experiment. The model misclassifies an extremely dark roof, which is much darker than the roofs in the training set, as a paved area. This could lead to a dangerous landing, potentially fatal for the drone, as the roof appears to be steep. This error underscores the training set's lack of generalization and suggests that including pixel-level transformations that adjust brightness and contrast might improve the model's performance on darker images and roofs.

VII. Conclusions

Through this project, we aimed to enhance scene understanding and identify safe landing zones in urban environments, utilizing advanced deep learning techniques. Through the implementation of semantic segmentation

techniques, we aimed to develop models that can accurately label every pixel in an aerial image, distinguishing between various objects and terrains, which is crucial for the safe operation of drones in diverse environments. After that, we developed a simple but effective algorithm to find the best spot for landing.

Our exploration of different neural network architectures yielded interesting results. Each model demonstrated unique strengths and limitations, but the fine-tuned UNet consistently emerged as the top performer, achieving the highest mean Intersection over Union of 58.01% on the test set. This model excelled at accurately segmenting difficult classes, outperforming both the manually defined UNet and the fine-tuned PSPNet across all evaluation metrics used. We also found that using data augmentation slightly improved all models' performance, showing how important this technique is for training strong segmentation networks, even with limited training data.

Observational evaluations of segmented images revealed that the fine-tuned UNet provided the most precise and detailed segmentation, especially in scenarios with complex urban features and mixed vegetation. While the PSPNet offered smoother outputs that might be useful in some contexts, it often missed finer details, a limitation that might be attributed to its scene parsing approach, however it's training was slightly faster, given the smaller number of trainable parameters. On the other hand, the manually defined UNet, despite having lower overall metrics, sometimes produced visually consistent segmentations.

However, the segmentation task also revealed areas for potential improvement. Due to computational limitations, we had to resize images to 256x256 pixels and only used spatial-level data augmentation. Future enhancements could involve using more powerful computational resources to maintain larger image sizes or even trying a patch-based approach. We could also explore various augmentation strategies, including pixel-level transformations, which could greatly help generalize the model's capabilities. Additionally, there are opportunities to experiment with other state-of-the-art architectures like ResNet or DeepLabV3+.

Moving beyond segmentation, our algorithm for drone landing detection proved highly effective. By processing segmented images, it autonomously identified safe landing spots by combining distance transform maps, ensuring drones avoid obstacles and stay away from image borders. This approach consistently identified optimal landing spots in test scenarios, underscoring its reliability.

Nevertheless, a notable limitation observed was the misclassification of roofs covered in gravel as safe landing zones due to dataset constraints. This issue highlights the importance of dataset diversity and generalization for real-world applications. Future directions could involve testing the model on live drone footage to evaluate its real-time performance and assess computational efficiency in processing multiple frames per second.

In conclusion, while our semantic segmentation models and landing detection algorithm have demonstrated promising capabilities, further advancements in model architecture, dataset diversity, and real-time application testing are crucial for improving the robustness of autonomous drone technologies in complex urban environments.

References

[1] I. of Computer Graphics and G. U. o. T. Vision, "Semantic Drone Dataset." 2019.

[2] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," *Information*, vol. 11, no. 2, 2020, doi: 10.3390/info11020125.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, 2015, pp. 234–241.

[4] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[5] P. Iakubovskii, "Segmentation Models Pytorch." GitHub, 2019.

[6] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035, 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.

[8] M. Everingham, L. Van~Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results."

[9] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[10] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.