Where to land your drone

Daniele Cecca
Matr. 914358
MSc Artificial Intelligence for Science and Technology
Email: d.cecca@campus.unimib.it

***Abstract*—This study presents the development of an intelligent system designed to analyze drone-captured images and identify the safest and most suitable landing spot. The system is structured into three key components.**

First, the dataset is examined, with a focus on the techniques that have been used to augment the data. Next, advanced image segmentation techniques are applied to accurately distinguish between different zones. Several state-of-the-art models (Unet, ResUnet, SegFormer, DeeplabV3+) are designed,implemented and compared based on some performance metrics . Finally, a post-processing pipeline is introduced to identify the optimal landing zone from the segmented image using clustering. The results demonstrate the system's effectiveness in providing reliable landing spot detection.

1. Introduction

Autonomous drones are increasingly used in a wide range of real-world applications, from aerial photography and environmental monitoring to delivery services and emergency response. While many advances have been made in path planning and navigation, one critical challenge remains underexplored: identifying safe and reliable landing zones in complex and unstructured environments. The ability of a drone to autonomously select an optimal landing spot is essential not only for completing missions efficiently but also for ensuring safety during emergency situations or battery constraints.

This work addresses that challenge by proposing a complete and intelligent system that processes drone-captured images to determine the best landing area based on semantic understanding of the scene. At the core of this system lies a semantic segmentation pipeline capable of classifying each pixel in an image into meaningful categories—such as paved surfaces, vegetation, water, and obstacles. This segmentation step enables the drone to "understand" the environment in human-like terms, thereby informing better landing decisions.

To achieve this, we explore and compare several state-of-the-art deep learning models tailored for semantic segmentation: U-Net, ResUNet, SegFormer, and DeepLabV3+. Each architecture is implemented, trained, and evaluated under the same experimental conditions, with special attention given to hyperparameter tuning, data augmentation, and optimization techniques. Given the relatively small size of the Semantic Drone Dataset used in this study, various preprocessing strategies are applied to increase diversity and reduce overfitting.

Beyond segmentation, a post-processing step is introduced to transform the pixel-wise predictions into actionable landing decisions. This involves mapping semantic labels to numerical scores based on landing suitability and applying hierarchical clustering to identify the most promising regions. The cluster with the highest average score is selected as the optimal landing zone, ensuring both spatial coherence and semantic relevance.

By combining semantic segmentation with unsupervised clustering, this system provides a complete and adaptable solution for autonomous drone landing. The results highlight the reliability of each model in producing meaningful predictions, and the effectiveness of the scoring and clustering strategy in selecting safe landing spots. This research not only demonstrates the practical utility of combining vision and learning techniques in robotics but also lays the groundwork for future enhancements in drone autonomy and safety.

2. Methodolgies

2.1. Semantic Segmentation

Semantic image segmentation (SiS) is a computer vision task in which the goal is to determine the semantic class of each pixel in an image. This task is typically approached in a supervised learning setting, relying on a dataset of pixel-level annotated images to train a machine learning model.

Like is written in [10]we can categorize image segmentation methods into two main classes:

- **Generic Image Segmentation (GIS)**
- **Promptable Image Segmentation (PIS)**

Generic Image Segmentation (GIS).. GIS methods can be further divided into: (i)

1) **Semantic segmentation**, which aims to identify and label each pixel with a semantic class from a predefined set $\mathcal{C}$.
2) **Instance segmentation**, which groups pixels that belong to the same semantic class into separate object instances.

3) **Panoptic segmentation**, which combines semantic and instance segmentation by predicting per-pixel class and instance labels, thus providing a comprehensive scene understanding.

Promptable Image Segmentation (PIS).. PIS extends GIS by incorporating a set of prompts $\mathcal{P}$ that specify the target to segment. In general, PIS methods generate segmentation masks conditioned on these prompts, without necessarily predicting the class of the segmented regions. While the concept of "prompt" is relatively new in terminology, the underlying idea has been studied for many years.

Depending on the type of prompt, PIS methods can be grouped into the following categories:

1) **Interactive segmentation** aims to segment specific objects or parts based on user input (e.g., clicks, scribbles, bounding boxes, or polygons), with prompts $\mathcal{P} = \{\text{click}, \text{scribble}, \text{box}, \text{polygon}\}$ representing visual prompts.
2) **Referring segmentation** involves extracting regions corresponding to linguistic descriptions, with $\mathcal{P} = \{\text{linguistic phrase}\}$ representing textual prompts.
3) **Few-shot segmentation** targets the segmentation of novel objects in a query image using a few annotated support images, i.e., $\mathcal{P} = \{(\text{image}, \text{mask})\}$.

While significant progress has been made in these areas, prior works often address the different prompt types separately. In contrast, Foundation Model (FM)-based approaches aim to integrate all prompt types into a unified framework. Furthermore, *in-context segmentation* has recently emerged as a novel few-shot segmentation task.

3. Dataset

The dataset used is part of the Semantic Drone Dataset, available at http://dronedataset.icg.tugraz.at. It consists of 400 images, each paired with a corresponding semantic label mask and RGB mask.

(a) Data original

(b) Data mask rgb

Figure 1: Examples of dataset images

In total, the dataset includes 23 distinct semantic regions, listed below.

Name	R	G	B
unlabeled	0	0	0
paved-area	128	64	128
dirt	130	76	0
grass	0	102	0
gravel	112	103	87
water	28	42	168
rocks	48	41	30
pool	0	50	89
vegetation	107	142	35
roof	70	70	70
wall	102	102	156
window	254	228	12
door	254	148	12
fence	190	153	153
fence-pole	153	153	153
person	255	22	96
dog	102	51	0
car	9	143	150
bicycle	119	11	32
tree	51	51	0
bald-tree	190	250	190
ar-marker	112	150	146
obstacle	2	135	115
conflicting	255	0	0

TABLE 1: Semantic classes and their corresponding RGB values.

One major challenge encountered with this dataset is its

limited size: 400 images are typically insufficient to train a deep learning model effectively, particularly if the model is deep and complex. This results in limited data variance, making the bias–variance trade-off skewed towards high bias and high variance.

To mitigate this limitation, I applied various data augmentation techniques to increase data diversity and reduce overfitting.

3.1. Data pre-processing

3.1.1. Data resize. Initially, all images were resized to 256×256 pixels. This resizing was necessary because the original image resolution (4000×6000) was too large to be efficiently processed by the neural network, given memory and computational constraints.

Figure 2: Resized images

3.1.2. Data augmentation. To address the limited size of the dataset, I employed a range of data augmentation techniques to artificially increase the number of training samples.

The following augmentation techniques were applied:

- **Rotation**
- **Flipping**
- **Saturation adjustment**
- **Blurring (Gaussian blur)**
- **Gaussian noise**
- **Random cutout**[3]

These transformations were applied using a function that randomly selected and applied one or more augmentations from the above list to each image. This strategy effectively doubled the size of the dataset.

Despite this increase, I found the amount of training data still insufficient for deep learning. Therefore, I additionally applied **random cropping** on resized 1024×1024 images.

(a) Data augmented images (b) Data cropped

3.1.3. Data Normalization. After all these pre-processing steps I normalized the pixel values by dividing to 255.

4. Segmentation models

I implemented and tested several state-of-the-art models, among the ones in [2]. The goal was to identify the network that achieved the best performance on the given dataset. In particular, I designed and developed the following architectures:

- **UNet**[7]: A classical encoder–decoder architecture based on convolutional neural networks (CNNs). It is widely used for semantic segmentation tasks due to its effectiveness in learning from limited data.
- **ResUNet**: A variant of the UNet architecture where residual blocks are integrated into the encoder and decoder.
- **SegFormer**[9]: A simple, efficient yet powerful semantic segmentation framework which unifies Transformers with lightweight multilayer perception (MLP) decoders
- **DeepLabv3+**[1]: An advanced semantic segmentation model that uses Atrous Spatial Pyramid Pooling (ASPP) in the encoder to capture multi-scale context.

I decided to compare these networks because each one uses a different type of layer in both the encoder and decoder—such as convolutional, residual, transformer, and atrous convolution layers. I'm interested in observing how these architectural differences affect performance. All the models have roughly the same number of parameters and are lightweight in terms of model size.

4.0.1. U-Net. The UNet architecture used in this project follows the classical design originally proposed for biomedical image segmentation. It is composed of three main components: an encoder, a bottleneck, and a decoder. The main layers used are classical CNN, maxpooling, Convolutional transpose and skip connections.

4.0.2. ResU-Net. As a second network, I designed and developed a REST-UNet, which closely follows the architecture of the standard UNet. It consists of an encoder, a bottleneck, and a decoder. The main difference from the original UNet lies in the downsampling blocks: instead of using a double convolution, each block incorporates a residual block with a standard architecture. Following the ResNet design, the networks starts with a convolutional layer using a 7×7 kernel[4]. Additionally, I replaced batch normalization with instance normalization[8], due to the small batch size during training.

Residual blocks. are used to address the shattered gradient problem and to train deeper networks also there is evidence that both residual links and batch normalization make the loss surface smoother, which permits larger learning rates[6].

4.0.3. SegFormer. Subsequently, I designed and implemented a model inspired by the SegFormer architecture. The only difference lies in the attention mechanism: while SegFormer employs an efficient attention variant, my model uses the standard (classical) attention mechanism.

This network consists of an encoder based on hierarchical transformer blocks, which capture multi-scale contextual information. The decoder is a lightweight MLP that fuses features from different levels of the encoder and predicts the final semantic segmentation mask.

The main building blocks of the transformer-based architecture are as follows:

- Hierarchical Feature Representation
- Overlapping Patch Merging
- Mix-FFN

4.1. DeepLabV3+

The last network was DeepLabV3+, in this case I used the implementation at https://keras.io/examples/vision/deeplabv3_plus/ that used a pretrained cnn of resnet50 on Imagenet as backbone.

5. Train

5.0.1. HyperParameters Tuning. To find the best hyperparameters, I experimented with various values across different hyperparameters.

Hyperparameters:

- **epochs**: The number of times the entire training dataset is passed through the network.
- **learning rate**: The step size at each iteration while moving towards a minimum of the loss function.
- **drop out**: The fraction of input units to drop during training. Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero at each update during training.
- **batch size**: The number of training examples utilized in one iteration.
- **loss function**: The function that measures how well the model's predictions match the target values.
- **scheduler lr**: The mechanism for adjusting the learning rate during training, often to improve convergence and performance.
- **optimizer**: The algorithm used to update the weights of the neural network to minimize the loss function.

Specifically, I create a configuration that will be used by the Weights and Biases agent to set the different hyperparameters during various experiments. This allows us to systematically explore the effect of different hyper-parameter settings on the model's performance. In this case, I used the simplest agent which chose the combination of the parameters randomly.

Configuration:

Parameter	Values
epochs	100
learning rate	{0.001, 0.0001}
dropout	{0.3, 0.4, 0.5}
batch size	{1, 5, 10}
loss function	$\text{sparse}_c ategorical_c rossentropy$
optimizer	AdamW
scheduler	ReduceLROnPlatea

TABLE 2: Hyperparameter Configuration

5.0.2. Drop-out. Dropout randomly clamps a subset of hidden units to zero at each iteration of computation on the gradient. This makes the network less dependent on any given hidden unit and encourages the weights to have smaller magnitudes so that the change in the function due to the presence or absence of the hidden unit is reduced.

5.0.3. AdamW. Gradient descent with a fixed step size has a notable drawback: it tends to make large adjustments to parameters associated with large gradients, potentially overshooting optimal values, while making smaller adjustments to parameters with smaller gradients, possibly slowing down progress. This imbalance occurs because a single fixed learning rate is not tailored to the varying gradients across different parameters.

To address this issue, I opted to use AdamW (Adaptive Moment Estimation), a method that dynamically adjusts the learning rate for each parameter. AdamW computes individual adaptive learning rates based on estimates of the first and second moments of the gradients. It maintains two moving averages per parameter, effectively adapting the learning rate over time. This adaptive approach facilitates faster convergence by handling both steep and shallow gradients more effectively, thereby enhancing overall stability and performance of the optimization process.Also it adds weight decay, useful for regularization and to reduce bias.

5.0.4. Justification for hyperparameter choices. Hyperparameter tuning was limited to the learning rate, dropout rate and batch size due to model complexity.This choice ensured that the model remained within acceptable limits while exploring significant aspects that could impact performance. Additionally,fixed small batch sizes were used because the ADAMW optimizer can work more effectively with smaller batches and residual block makes the function more smooth. This approach benefits from more frequent updates, which can lead to better convergence properties and improved generalization.Also we have a limited amount of data and we are training data from scratch. All experiments and training were performed using a A100 GPU.

5.0.5. Initialization. Initializing the network parameters is critical. To avoid exploding gradient or vanishing gradient, I apply He initialization on most of the CNN layers.. The biases β are set to zero, and the weights Ω are chosen from a normal distribution with mean zero and variance $\frac{2}{D_h}$, where D_h is the number of hidden units in the previous layer.

5.1. Train U-Net

The agent run 5 different experiments, with five different combination of hyper-parameters, which are represented in the following picture.

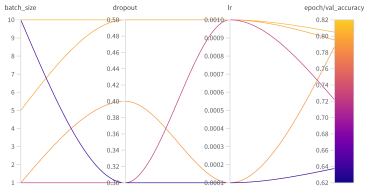

Figure 4: Different combination U-net

I trained all the model with 100 epochs and I evaluate them by computing the validation loss and the accuracy on the validation set. And here we can see a summary of all the behaviours of the training

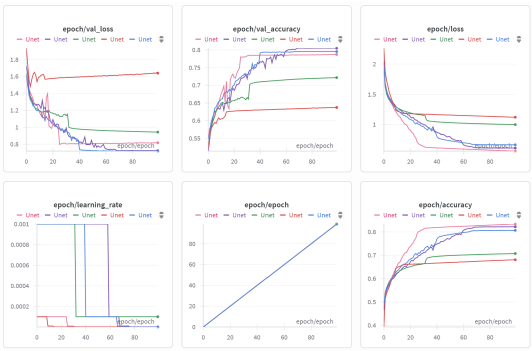

Figure 5: Training of 5 U-net

The model that performed the best has the following hyperparameters.

Parameter	Value
Batch Size	10
Dropout	0.5
Epochs	100
Learning Rate (lr)	0.001

We can observe that the best-performing model was the one with the highest dropout rate and the largest learning rate, which aligns perfectly with what we discussed in the hyperparameters section. A higher learning rate and dropout help explore the solution space more effectively and reduce bias.

The same training procedure was applied to the other networks.

5.2. Train ResU-Net

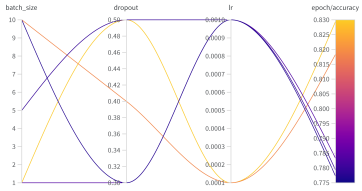

Figure 6: Different combination

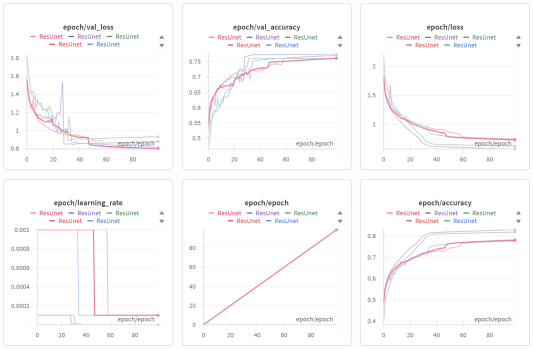

Figure 7: Training of 5 ResU-net

The model that performed the best was the following one:

Parameter	Value
Batch Size	1
Dropout	0.5
Epochs	10
Learning Rate (lr)	0.001

In the case of the ResNet, the results contrast with those of the U-Net. I believe this is due to the presence of residual layers, which allow each residual block to retain more information. As a result, the network can converge faster even with a smaller learning rate, leading to more efficient weight management. Additionally, I think the use of instance normalization also contributes to this improved performance.

5.3. Train Segformer

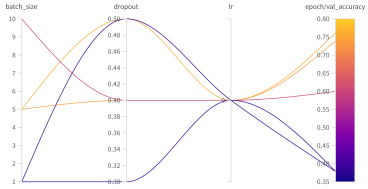

Figure 8: Different combination

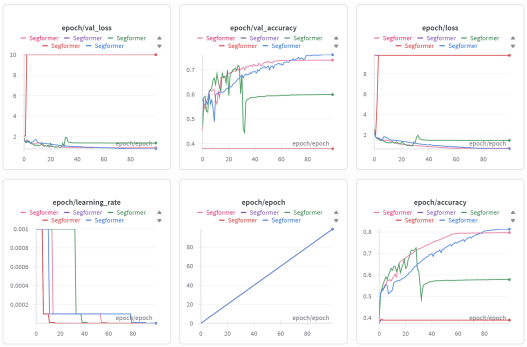

Figure 9: Training of 5 Seg-former

Best hyperparametrs:

Parameter	Value
Batch Size	5
Dropout	0.5
Epochs	100
Learning Rate (lr)	0.001

In this case, it was unlikely that a single learning rate would be optimal. We can also see that the worst result occurred with a low dropout rate.

5.4. Train DeepLabv3

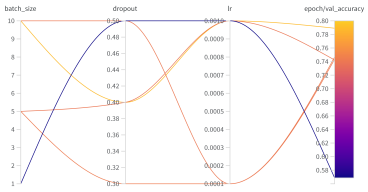

Figure 10: Different combination

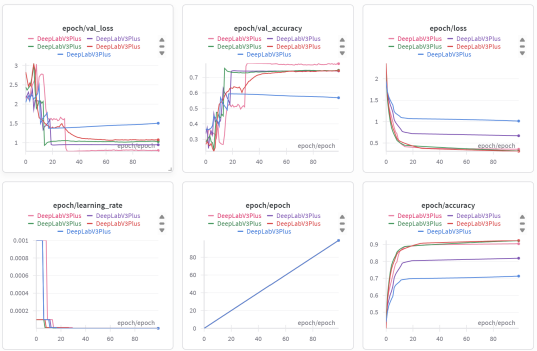

Figure 11: Training of 5 DeepVlabv3

Best hyperparameters:

Parameter	Value
Batch Size	10
Dropout	0.4
Epochs	100
Learning Rate (lr)	0.001

In this case we have observed the phenonema of the epoch-wise double descent. This phenomenon has been modeled by [5] in terms of different features in the model being learned at different speeds.

6. Test

For testing, I selected the best model of each type and evaluated them on the test set. Below is a summary table of the results.

TABLE 3: Performance comparison of segmentation models

Model	Accuracy	Precision (macro)	Recall (macro)	Mean Dice
UNet	0.828	0.560	0.458	0.532
ResUNet	0.802	0.457	0.432	0.479
Segformer	0.760	0.466	0.379	0.447
DeepLabV3	0.826	0.553	0.487	0.555

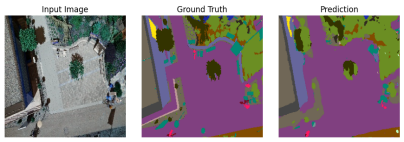

Figure 12: U-net output

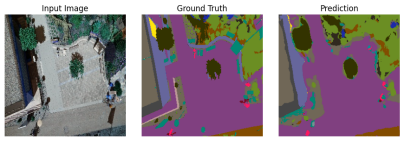

Figure 13: ResUnet output

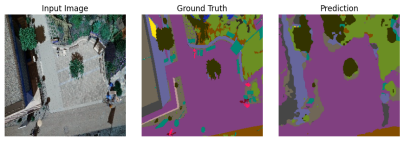

Figure 14: Segformer output

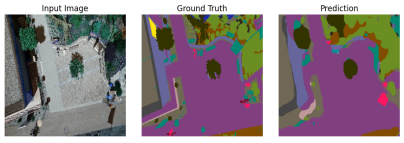

Figure 15: DeepLab output

7. Identifying the Best Landing Spot

To identify the most suitable landing spot for the drone, the following steps are performed:

1) A scoring function is defined based on the semantic class of each region.
2) Each pixel is mapped to a score using this function.
3) The resulting score matrix is clustered using a hierarchical agglomerative clustering algorithm.
4) The cluster with the highest average score is selected as the optimal landing spot.

Several approaches can be used for the scoring function, such as computing the Hamming distance or designing a custom metric. In this study, since the semantic labels are informative and meaningful, I chose to assign scores using a Gaussian function centered on the label corresponding to the best landing region: **paved-area**.

The scoring function is defined as follows:

$$\text{score}(x) = \exp\left(-\left(\frac{x-\mu}{2\sigma}\right)^2\right)$$

where x represents the label index of a semantic class, and μ corresponds to the label index of the *paved-area* class.

The normalized scores for each semantic class are shown below:

Semantic Class	Score
unlabeled	0.9948
paved-area	1.0000
dirt	0.9948
grass	0.9793
gravel	0.9541
water	0.9199
rocks	0.8777
pool	0.8288
vegetation	0.7744
roof	0.7161
wall	0.6553
window	0.5935
door	0.5319
fence	0.4718
fence-pole	0.4141
person	0.3597
dog	0.3092
car	0.2630
bicycle	0.2214
tree	0.1844
bald-tree	0.1521
ar-marker	0.1241
obstacle	0.1002
conflicting	0.0800

TABLE 4: Score assigned to each semantic class based on similarity to the ideal landing zone.

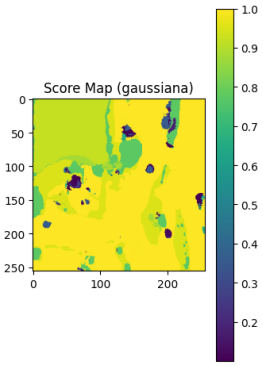

Figure 16: Different scoring combinations

To preserve spatial coherence, I apply agglomerative hierarchical clustering to the score matrix. I chose to generate 64 clusters so that each region corresponds to an area of at least 4×4 pixels, ensuring meaningful spatial granularity.

From the clustered regions, I select the top five clusters with the highest mean score as candidate landing spots.

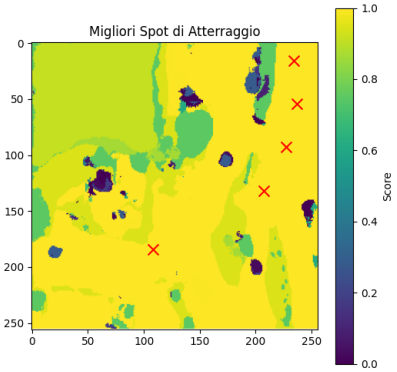

Figure 17: Top candidate landing regions after clustering

8. Conclusion

This study presented a modular and effective pipeline for detecting safe drone landing spots using semantic segmentation and spatial clustering techniques. Through systematic experimentation with multiple state-of-the-art segmentation models, including U-Net, ResUNet, SegFormer, and DeepLabV3+, all models demonstrated solid performance on the semantic drone dataset, each showing strengths depending on the metric considered. U-Net achieved the best overall accuracy, while DeepLabV3+ provided the highest Dice score, indicating that even with a limited dataset, the system was able to train robust and reliable segmentation models. The post-processing phase—based on a scoring function and hierarchical clustering—successfully identified optimal landing regions with semantic awareness and spatial coherence.

Overall, the results validate the feasibility of combining deep learning and unsupervised clustering to support autonomous drone navigation, laying a strong foundation for future improvements and applications in real-world scenarios.

References

[1] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[2] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. Semantic image segmentation: Two decades of research, 2023.

[3] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Mohammad Pezeshki, Amartya Mitra, Yoshua Bengio, and Guillaume Lajoie. Multi-scale feature learning dynamics: Insights for double descent, 2021.

[6] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[8] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[9] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.

[10] Tianfei Zhou, Wang Xia, Fei Zhang, Boyu Chang, Wenguan Wang, Ye Yuan, Ender Konukoglu, and Daniel Cremers. Image segmentation in foundation model era: A survey, 2024.

9. Disclosure Statement

The authors declare that this report is entirely original and does not contain any plagiarism.