SIGNAL AND IMAGING ACQUISITION AND MODELLING IN ENVIRONMENT - AERIAL DRONE IMAGE SEGMENTATION

Mirko Morello
920601
m.morello11@campus.unimib.it

Abstract

This study investigates automated methods for semantic segmentation of aerial drone imagery using UNET inspired convolutional neural networks (CNN). The research was motivated by the need for efficient ways to analyze large, high-resolution aerial datasets and produce precise segmentation maps for environmental monitoring and drone autopiloting systems.

The study utilized a dataset of high-resolution drone images with corresponding hand-labeled segmentation masks. To handle the large image sizes, a patch-based approach was employed, dividing images into smaller patches that can be processed independently by the CNNs employed. A multi-scale "stitch level" method was introduced to capture both fine details and broader context. Extensive data augmentation, including color variations, noise, and blur, was applied to improve model robustness.

Two state-of-the-art CNN architectures were evaluated: Resnet34 and EfficientNet-B5, both as backbones of DeepLabV3+, as weel as a custom made model. Careful experiments were conducted to determine optimal patch sizes and receptive field. The models achieved impressive results, with Resnet34 slightly outperforming EfficientNet-B5 and the custom model. The patch and multi-scale stitch level approach proved crucial for maximizing performance.

1 INTRODUCTION

Aerial drone imagery has become an invaluable tool for environmental monitoring, urban planning, precision agriculture, and many other applications. The ability to efficiently and accurately analyze high-resolution aerial images to produce semantic segmentation maps is crucial for extracting actionable insights from this data. However, manual annotation of such large-scale datasets is prohibitively time-consuming and labor-intensive. In recent years, deep learning approaches, particularly convolutional neural networks (CNNs), have achieved remarkable success in semantic segmentation tasks across various domains. CNNs have the ability to automatically learn hierarchical features from raw image data, enabling them to produce dense, pixel-wise classifications. This makes them well-suited for processing complex, high-resolution aerial imagery.

However, applying CNNs to very high-resolution drone imagery poses several challenges. The large image sizes make it computationally infeasible to process entire images at once, necessitating patch-based or sliding window approaches. The choice of patch size involves a trade-off between capturing fine details and understanding broader context. Extremely high-resolution data also tends to exhibit large variations in object scale and appearance, requiring models to be robust and capable of capturing multi-scale features.

This study aims to address these challenges by investigating optimal patch-based CNN architectures and training methodologies for semantic segmentation of high-resolution aerial imagery. We introduce a novel multi-scale "stitch level" approach to capture both fine details and broader context, and employ extensive data augmentation to improve model robustness. Our goal is to develop an efficient and accurate system for automated analysis of large-scale aerial datasets, enabling more effective environmental monitoring and land use mapping.

2 DATA AND METHODOLOGY

2.1 Dataset

The dataset used in this study consists of 400 high-resolution aerial images captured by drones, with dimensions of 6000×4000 pixels. Each image is accompanied by a carefully hand-labeled segmentation mask, providing pixel-wise ground truth labels for training and evaluation of segmentation models, an example can be seen in Figure 1. The images cover a diverse range of environments, including urban areas and rural landscapes. This diversity is crucial for developing models that can generalize well to different scenarios encountered in real-world applications.

The masks provided feature 24 unique labels, which asssign each pixel to its own category, which are the following:

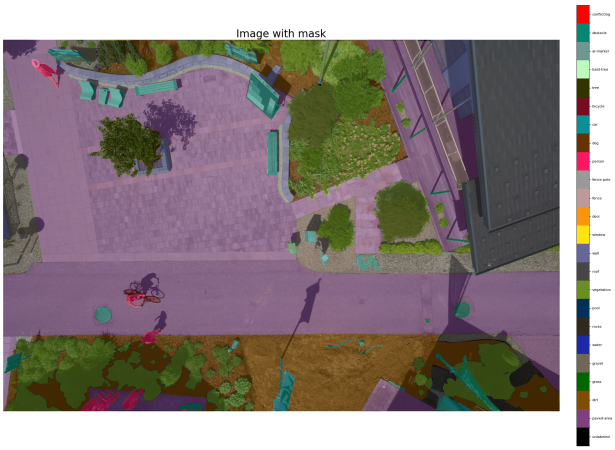

Figure 1: Example of image with its corresponding mask overlapped and with 50% opacity.

- unlabeled
- paved-area
- dirt
- grass
- gravel
- water
- rocks
- pool
- vegetation
- roof
- wall
- window
- door
- fence
- fence-pole
- person
- dog
- car
- bicycle
- tree
- bald-tree
- ar-marker
- obstacle
- conflicting

Without any doubt, the main core and focus of the entire project is being able to handle correctly this challenging dataset, and this is where most of the work has been done.

2.2 Patches methodology

Processing such large, high-resolution images with CNNs poses significant computational challenges. To address this, we adopt a patch-based approach, where each image is divided into smaller, overlapping patches that can be processed independently. This allows for more efficient use of the data we have at our disposal.
Choosing an appropriate patch size is crucial to capture relevant contextual information while maintaining computational efficiency. A larger patch size increases the receptive field, allowing the CNN to capture broader context, but it also increases computational complexity. On the other hand, a smaller patch size reduces computational demands but may limit the CNN's ability to capture long-range dependencies [10].
We experiment with various patch sizes to find the optimal balance for our task.

2.2.1 Multi-scale Stitch Levels

To further enhance the model's ability to capture multi-scale features, we introduce a novel "stitch level" approach. This involves generating patches at multiple scales for each image, allowing the model to simultaneously process both fine details and larger patterns. By processing patches at multiple scales, the model can learn to integrate information from different receptive fields, leading to improved segmentation performance [1].
Formally, let I be the input image, P be the base patch size, and L be the number of stitch levels. For each level $\ell \in [0, L-1]$, we generate patches of size:

$$P_\ell = P \cdot 2^\ell$$

These multi-scale patches are then resized to the base patch size P before being fed into the transform pipeline like the other patches, ensuring a consistent input size while preserving multi-scale information, an example of a final processed image can be seen in Figure 2.

The dataset has been split into train and test sets with a ratio of 80:20 respectively.

2.3 Data augmentation

To improve model robustness and generalization, we apply extensive data augmentation during training. In our augmentation pipeline, implemented using the Albumentations library, we pondered the following:

- Geometric transformations: rotations, flips, scaling, translation
- Color variations: changes in brightness, contrast, hue, saturation
- Noise injection: Gaussian noise, blur
- Geometrical distortions: Elastic Transformation, Grid Distortion, Optical Distortion

Crucially, identical transformations are applied to both the input patch and its segmentation mask to maintain alignment, this ensures that the spatial relationship between image features and their segmentation labels is preserved throughout the augmentation process.
The augmentation pipeline was carefully tuned to balance the diversity of transformations with computational efficiency. Augmenting individual patches instead of entire images effectively increased the number of augmented samples. However, this led to significant bottlenecks between the GPU and CPU during training, with the GPU waiting for the CPU to prepare the batches. To address this, we removed the computationally intensive geometrical distortion transformations from the pipeline. This optimization reduced the waiting time for the GPU, resulting in faster training iterations and improved model

Figure 2: Visual comparison of the original image and its reconstructed version using augmented patches. The left panel displays the original aerial image, while the right panel presents the reconstructed image composed of individually processed patches. The patches have undergone data augmentation, including random flips and rotations, making it challenging to perceive their direct correspondence to the original image. The last five patches in the reconstructed image are the result of the multi-scale stitch level approach, where larger portions of the image are captured at different scales.

convergence, while still maintaining a diverse set of augmentations to enhance the model's robustness and generalization capabilities.
We used a probabilistic approach, where each transformation has a certain probability of being applied to any given image, ensuring a wide variety of augmented samples without overly distorting the original data distribution.

As part of the data augmentation pipeline, normalization of all the patches has been applied to standardize the input data. We used the normalization values commonly employed when working with models pre-trained on the ImageNet dataset [14]. The mean values (0.485, 0.456, 0.406) and standard deviation values (0.229, 0.224, 0.225) were used to center and scale the pixel values of the patches. These normalization values are derived from the statistics of the ImageNet dataset, which is widely used for pre-training deep learning models. It is important to note that while other data augmentation transformations were applied only to the training set to introduce variability and improve model generalization, normalization was applied to both the training and test sets. This ensures that the input data is consistently scaled and centered across both sets, facilitating fair evaluation and comparison of the model's performance.
At the end of the data augmentation pipeline, all the patches are resized to 256×256, regardless of their initial size.

2.4 Segmentation Architectures

In this study, we employ the DeepLabV3+ architecture [2] with different backbone architectures, and a custom made model. DeepLabV3+ is a state-of-the-art model that combines the strengths of atrous convolutions [19], spatial pyramid pooling, and encoder-decoder structure to achieve high-quality segmentation results. The DeepLabV3+ architecture consists of two main components: an encoder and a decoder. The encoder is responsible for extracting multi-scale contextual information from the input image, while the decoder refines the segmentation map by combining the encoded features with low-level features from earlier layers.

The integration of the backbone network into the DeepLabV3+ architecture is seamless. The backbone network is used as the encoder, and its output features are processed by atrous convolutions and spatial pyramid pooling to capture multi-scale contextual information. The decoder then combines these encoded features with low-level features from earlier layers of the backbone network to produce a refined segmentation map. To leverage the pre-trained weights of the backbone networks, we initialize them with weights trained on the ImageNet dataset [3]. This transfer learning approach allows us to benefit from the knowledge acquired by the backbone networks on a large-scale image classification task and fine-tune them for our specific aerial image segmentation task.
The choice of backbone architecture plays a crucial role in the performance and efficiency of the DeepLabV3+ model. ResNet34 offers a deeper network with skip connections, enabling the capture of complex features and spatial relationships. EfficientNet-B5, on the other hand, provides a parameter-efficient alternative that can achieve good performance with fewer parameters. By experimenting with these different backbone architectures and our custom made model, we aim to find the optimal balance between segmentation accuracy and computational efficiency for our specific use case.

It is worth noting that both the custom network and the pre-trained models faced challenges with batch sizes due to memory constraints. To mitigate this issue, we implemented gradient accumulation, a technique that simulates

larger batch sizes by accumulating gradients over multiple iterations before updating the model parameters [7, 12]. Gradient accumulation allows for effective training with increased effective batch sizes while working within the available memory limits. However, it comes at the cost of increased training time and potential instability in the optimization process.

To ensure proper batch normalization statistics [18, 8, 15] in the presence of gradient accumulation, we developed a novel approach to accumulate and update these statistics across multiple iterations. This was achieved by maintaining separate accumulators for the running mean and variance of each batch normalization layer and updating them incrementally as gradients were accumulated. Once the desired number of accumulation steps was reached, the accumulators were used to update the global running statistics of the batch normalization layers. This technique enabled accurate batch normalization calculations even with gradient accumulation, contributing to the robustness and reliability of our training process.

On the other hand, the lighter architecture of ResNet34 allowed us to employ a larger batch size during training compared to our custom network, without the necessity of gradient accumulation. This is a significant advantage, as previous research has demonstrated that excessively small batch sizes can lead to suboptimal generalization performance [9, 11, 4], and the gradient accumulation technique mitigates it but not fully.

2.4.1 ResNet34 as Backbone for DeepLabV3+

ResNet34 [5] is a 34-layer deep residual network that introduces skip connections to mitigate the vanishing gradient problem and enable the training of deeper networks. By allowing the network to learn residual functions, ResNet34 can capture rich hierarchical features at different levels of abstraction. When used as the backbone in DeepLabV3+, ResNet34 provides a strong foundation for extracting meaningful features from the input aerial images, the architecture can be seen in Table 1.

2.4.2 EfficientNet-B5 as Backbone for DeepLabV3+

EfficientNet-B5 [16] is a highly parameter-efficient CNN architecture that achieves excellent performance while maintaining a relatively low number of parameters. Despite the name, it is the heaviest and most complex architecture in our study. It employs a compound scaling method that uniformly scales the network's depth, width, and resolution. EfficientNet-B5 has shown impressive results on image classification tasks and has the potential to serve as an effective backbone for semantic segmentation. Its use in combination with DeepLabV3+ has been already proposed for image segmentation as a solution for high-profile missions, such as a scene area judgment for the Mars Unmanned Vehicle System [6]. The architecture can be seen in Table 2

Layer	Details	Output Shape
Input Image: $3 \times 256 \times 256$		
conv1	Conv2d(3, 64, 7x7, stride=2, padding=3) BatchNorm2d(64) ReLU(inplace=True) MaxPool2d(3x3, stride=2, padding=1)	$64 \times 128 \times 128$ $64 \times 128 \times 128$ $64 \times 128 \times 128$ $64 \times 64 \times 64$
layer1	BasicBlock Conv2d(64, 64, 3x3, stride=1, padding=1) BatchNorm2d(64) ReLU(inplace=True) Conv2d(64, 64, 3x3, stride=1, padding=1) BatchNorm2d(64)	$64 \times 64 \times 64$
layer2	BasicBlock Conv2d(64, 128, 3x3, stride=2, padding=1) BatchNorm2d(128) ReLU(inplace=True) Conv2d(128, 128, 3x3, stride=1, padding=1) BatchNorm2d(128) Downsample: Conv2d(64, 128, 1x1, stride=2)	$128 \times 32 \times 32$
layer3	BasicBlock Conv2d(128, 256, 3x3, stride=2, padding=1) BatchNorm2d(256) ReLU(inplace=True) Conv2d(256, 256, 3x3, stride=1, padding=1) BatchNorm2d(256) Downsample: Conv2d(128, 256, 1x1, stride=2)	$256 \times 16 \times 16$
layer4	BasicBlock Conv2d(256, 512, 3x3, stride=1, padding=2, dilation=2) BatchNorm2d(512) ReLU(inplace=True) Conv2d(512, 512, 3x3, stride=1, padding=2, dilation=2) BatchNorm2d(512) Downsample: Conv2d(256, 512, 1x1, stride=1, dilation=2)	$512 \times 16 \times 16$
Decoder		
ASPP	ASPP Conv2d(512, 256, 1x1) BatchNorm2d(256) ReLU(inplace=True) SeparableConv2d(512, 256, 3x3, padding=12, dilation=12) BatchNorm2d(256) ReLU(inplace=True)	$256 \times 16 \times 16$
up	UpsamplingBilinear2d(scale factor=4.0)	$256 \times 64 \times 64$
block1	Conv2d(64, 48, 1x1)	$48 \times 64 \times 64$
block2	SeparableConv2d(304, 256, 3x3, padding=1) BatchNorm2d(256) ReLU(inplace=True)	$256 \times 64 \times 64$
Segmentation Head		
Conv2d	Conv2d(256, 24, 1x1)	$24 \times 64 \times 64$
upsampling	UpsamplingBilinear2d(scale factor=4.0)	$24 \times 256 \times 256$
Activation	Identity()	$24 \times 256 \times 256$

Table 1: DeepLabV3Plus Resnet34 backbone Architecture. 22,443,368 parameters.

Layer	Details	Output Shape
Input Image: $3 \times 256 \times 256$		
conv_stem	Conv2dStaticSamePadding(3, 48, 3x3, stride=2, padding=1) BatchNorm2d(48)	$48 \times 128 \times 128$ $48 \times 128 \times 128$
enc1	MBConvBlock (expanded 1x; repeated 2x)	$24 \times 128 \times 128$
enc2	MBConvBlock (expanded 6x; repeated 4x)	$40 \times 64 \times 64$
enc3	MBConvBlock (expanded 6x; repeated 4x)	$64 \times 32 \times 32$
enc4	MBConvBlock (expanded 6x; repeated 12x)	$176 \times 16 \times 16$
bridge	MBConvBlock (expanded 6x, dilated 2x; repeated 3x)	$512 \times 16 \times 16$
aspp	Conv2d(512, 256, 1x1, stride=1) + BatchNorm2d + ReLU SeparableConv2d(512, 256, 3x3, stride=1, padding=d, dilation=d) $d \in \{12, 24, 36\}$, repeated for each dilation rate	$256 \times 16 \times 16$
aspp_pool	AdaptiveAvgPool2d(output_size=1) + Conv2d(512, 256, 1x1)	$256 \times 1 \times 1$
aspp_out	Conv2d(1280, 256, 1x1, stride=1) + BatchNorm2d + ReLU + Dropout(p=0.5)	$256 \times 16 \times 16$
dec1	SeparableConv2d(256, 256, 3x3, stride=1, padding=1) + BatchNorm2d + ReLU	$256 \times 16 \times 16$
up	UpsamplingBilinear2d(scale_factor=4.0)	$256 \times 64 \times 64$
enc1_reduced	Conv2d(24, 48, 1x1, stride=1) + BatchNorm2d + ReLU	$48 \times 128 \times 128$
dec2	SeparableConv2d(304, 256, 3x3, stride=1, padding=1) + BatchNorm2d + ReLU	$256 \times 128 \times 128$
final_conv	Conv2d(256, 24, 1x1, stride=1)	$24 \times 128 \times 128$
final_up	UpsamplingBilinear2d(scale_factor=2.0)	$24 \times 256 \times 256$
output	Activation(Identity())	$24 \times 256 \times 256$

Table 2: DeepLabV3Plus with EfficientNet-B5 encoder architecture. The table has been concentrated due to the large number of layers. 28,445,656 parameters.

2.4.3 Custom Made Model

In addition to the DeepLabV3+ architecture with different backbones, we develop a custom segmentation model that draws inspiration from the U-Net architecture [13], the DeepLab model [1, 2], and attention mechanisms [17]. The architecture of this custom model is presented in Table 3.

The custom model adopts an encoder-decoder structure with skip connections, reminiscent of the U-Net architecture. The encoder blocks comprise convolutional layers, batch normalization, and ReLU activation, along with attention blocks that capture long-range dependencies and focus on relevant features. The decoder blocks aim to recover spatial resolution and refine the segmentation output by employing upsampling, skip connections, and attention blocks. A central component of the custom model is the Atrous Spatial Pyramid Pooling (ASPP) module [1], inspired by the DeepLab model. The ASPP module utilizes atrous convolutions with varying dilation rates to capture multi-scale contextual information, enhancing the model's ability to handle objects of different sizes and improving its segmentation performance.

Furthermore, the custom model incorporates attention mechanisms, drawing from the self-attention concept used in models like the Transformer [17]. The attention blocks learn to assign importance to different spatial locations in the feature maps, enabling the model to focus on relevant features and enhance its discriminative power.

The combination of the encoder-decoder structure with skip connections, the ASPP module, and attention blocks allows the custom model to effectively extract features, recover spatial resolution, and capture multi-scale context for improved segmentation performance.

Despite the low number of parameters, the architecture is not as efficient as the other state of the art models, which resulted in relatively shallow layers and simple architecture.

Layer	Details	Output Shape
	Input Image: $3 \times 256 \times 256$	
enc1	Conv2d(3, 32, 3x3, stride=1, padding=1)	$32 \times 256 \times 256$
	BatchNorm2d(32)	$32 \times 256 \times 256$
	Conv2d(32, 32, 3x3, stride=1, padding=1)	$32 \times 256 \times 256$
	BatchNorm2d(32)	$32 \times 256 \times 256$
	ReLU(inplace=True)	$32 \times 256 \times 256$
	MaxPool2d(2x2, stride=2)	$32 \times 128 \times 128$
	AttentionBlock(Conv2d(32, 1, 1x1, stride=1), Sigmoid())	$32 \times 128 \times 128$
enc2	Conv2d(32, 64, 3x3, stride=1, padding=1)	$64 \times 128 \times 128$
	BatchNorm2d(64)	$64 \times 128 \times 128$
	Conv2d(64, 64, 3x3, stride=1, padding=1)	$64 \times 128 \times 128$
	BatchNorm2d(64)	$64 \times 128 \times 128$
	ReLU(inplace=True)	$64 \times 128 \times 128$
	MaxPool2d(2x2, stride=2)	$64 \times 64 \times 64$
	AttentionBlock(Conv2d(64, 1, 1x1, stride=1), Sigmoid())	$64 \times 64 \times 64$
enc3	Conv2d(64, 128, 3x3, stride=1, padding=1)	$128 \times 64 \times 64$
	BatchNorm2d(128)	$128 \times 64 \times 64$
	Conv2d(128, 128, 3x3, stride=1, padding=1)	$128 \times 64 \times 64$
	BatchNorm2d(128)	$128 \times 64 \times 64$
	ReLU(inplace=True)	$128 \times 64 \times 64$
	MaxPool2d(2x2, stride=2)	$128 \times 32 \times 32$
	AttentionBlock(Conv2d(128, 1, 1x1, stride=1), Sigmoid())	$128 \times 32 \times 32$
enc4	Conv2d(128, 256, 3x3, stride=1, padding=1)	$256 \times 32 \times 32$
	BatchNorm2d(256)	$256 \times 32 \times 32$
	Conv2d(256, 256, 3x3, stride=1, padding=1)	$256 \times 32 \times 32$
	BatchNorm2d(256)	$256 \times 32 \times 32$
	ReLU(inplace=True)	$256 \times 32 \times 32$
	MaxPool2d(2x2, stride=2)	$256 \times 16 \times 16$
	AttentionBlock(Conv2d(256, 1, 1x1, stride=1), Sigmoid())	$256 \times 16 \times 16$
bridge	Conv2d(256, 512, 1x1, stride=1)	$512 \times 16 \times 16$
	Conv2d(256, 512, 3x3, stride=1, padding=6, dilation=6)	$512 \times 16 \times 16$
	Conv2d(256, 512, 3x3, stride=1, padding=12, dilation=12)	$512 \times 16 \times 16$
	Conv2d(256, 512, 3x3, stride=1, padding=18, dilation=18)	$512 \times 16 \times 16$
	AdaptiveAvgPool2d(output_size=(1, 1))	$512 \times 1 \times 1$
	Conv2d(256, 512, 1x1, stride=1)	$512 \times 1 \times 1$
	Conv2d(2560, 512, 1x1, stride=1)	$512 \times 16 \times 16$
	BatchNorm2d(512)	$512 \times 16 \times 16$
	ReLU(inplace=True)	$512 \times 16 \times 16$
dec4	ConvTranspose2d(512, 256, 2x2, stride=2)	$256 \times 32 \times 32$
	Conv2d(512, 256, 3x3, stride=1, padding=1)	$256 \times 32 \times 32$
	BatchNorm2d(256)	$256 \times 32 \times 32$
	Conv2d(256, 256, 3x3, stride=1, padding=1)	$256 \times 32 \times 32$
	BatchNorm2d(256)	$256 \times 32 \times 32$
	ReLU(inplace=True)	$256 \times 32 \times 32$
	AttentionBlock(Conv2d(256, 1, 1x1, stride=1), Sigmoid())	$256 \times 32 \times 32$
dec3	ConvTranspose2d(256, 128, 2x2, stride=2)	$128 \times 64 \times 64$
	Conv2d(256, 128, 3x3, stride=1, padding=1)	$128 \times 64 \times 64$
	BatchNorm2d(128)	$128 \times 64 \times 64$
	Conv2d(128, 128, 3x3, stride=1, padding=1)	$128 \times 64 \times 64$
	BatchNorm2d(128)	$128 \times 64 \times 64$
	ReLU(inplace=True)	$128 \times 64 \times 64$
	AttentionBlock(Conv2d(128, 1, 1x1, stride=1), Sigmoid())	$128 \times 64 \times 64$
dec2	ConvTranspose2d(128, 64, 2x2, stride=2)	$64 \times 128 \times 128$
	Conv2d(128, 64, 3x3, stride=1, padding=1)	$64 \times 128 \times 128$
	BatchNorm2d(64)	$64 \times 128 \times 128$
	Conv2d(64, 64, 3x3, stride=1, padding=1)	$64 \times 128 \times 128$
	BatchNorm2d(64)	$64 \times 128 \times 128$
	ReLU(inplace=True)	$64 \times 128 \times 128$
	AttentionBlock(Conv2d(64, 1, 1x1, stride=1), Sigmoid())	$64 \times 128 \times 128$
dec1	ConvTranspose2d(64, 32, 2x2, stride=2)	$32 \times 256 \times 256$
	Conv2d(64, 32, 3x3, stride=1, padding=1)	$32 \times 256 \times 256$
	BatchNorm2d(32)	$32 \times 256 \times 256$
	Conv2d(32, 32, 3x3, stride=1, padding=1)	$32 \times 256 \times 256$
	BatchNorm2d(32)	$32 \times 256 \times 256$
	ReLU(inplace=True)	$32 \times 256 \times 256$
	AttentionBlock(Conv2d(32, 1, 1x1, stride=1), Sigmoid())	$32 \times 256 \times 256$
final	Conv2d(64, 24, 1x1, stride=1)	$24 \times 256 \times 256$

Table 3: Custom model architecture. 9,341,600 parameters.

2.4.4 Training Details

Models are trained using the AdamW optimizer with a learning rate of 1e-4 and weight decay of 1e-4. We employ a cosine annealing learning rate schedule over 20 epochs for the various experiments, and 100 epochs for the final most promising model.

For the segmentation loss, we experimented with cross-entropy, and two different custom made losses: Dice and Focal loss. Upon not noticing any significant gain using these losses after the first experiments, we decided to use cross-entropy loss for consistency, as running all the experiments with all the possible combination of losses was unfeasible.

We monitor performance on a held-out test set during training, and save the model checkpoint with the lowest validation loss score.

2.5 Evaluation Metrics

We employ several standard metrics to comprehensively evaluate the performance of our semantic segmentation models:

- **Mean Intersection-over-Union (mIoU)**: mIoU is the primary evaluation metric, providing a balanced measure of segmentation quality across all classes. It is calculated by taking the intersection of the predicted and ground truth masks for each class, dividing by their union, and averaging across all classes. mIoU is sensitive to both false positives and false negatives, making it a robust indicator of overall segmentation accuracy.

- **Dice Score**: The Dice score, also known as the F1 score, is another common metric for segmentation tasks. It is computed as the harmonic mean of precision and recall, where precision is the percentage of true positive pixels among all predicted positives,

and recall is the percentage of true positives among all actual positives.

- **Accuracy**: Pixel accuracy measures the percentage of pixels correctly classified across all classes. While a straightforward metric, accuracy can be misleading for imbalanced datasets where the background class dominates, therefore, we report accuracy alongside other metrics for a more comprehensive evaluation.

- **Per-class IoU**: In addition to mIoU, we calculate the IoU for each individual class. Per-class IoU scores provide insights into the model's performance on specific categories, helping identify classes that the model struggles with or excels at.

- **Inference Speed (FPS)**: To assess the efficiency of our models for real-time applications, we measure the inference speed in frames per second (FPS) on a desktop GPU (Nvidia RTX 3090). An high FPS could allow atonomous driving or landing of the drone itself, or real time environment segmentation supervisioned by a human operator.

By considering this combination of metrics - mIoU, Dice score, accuracy, per-class IoU, and FPS - we aim to comprehensively evaluate our semantic segmentation models in terms of both segmentation quality and computational efficiency.

3 Results

3.1 Impact of Patch Size

We conduct experiments with different patch sizes (500, 1000, 2000) to study the impact of receptive field on segmentation quality. Table 4 shows the mIoU scores achieved by DeepLabV3+ with Resnet34 with each patch size after 20 epoches. It is important to notice that having a smaller patch size produces more images, thus reducing the size of the batch, this in turn has an effect on the results, as discussed previously.

Size	Patches	Patches with max stitching	Accuracy
500	96	127	0.8483
1000	24	31	0.8973
2000	4	5	0.8654

Table 4: Number of total patches per image when varying the patch size, the stitching level considered is the highest it can be achieved for that size.

Results show that a patch size of 1000 provides the best balance between capturing fine details and understanding broader context. Smaller patches (500) struggle to capture sufficient context, while larger patches (2000) lead to overly coarse predictions and loss of fine details, a comparison can be seen in picture 3.

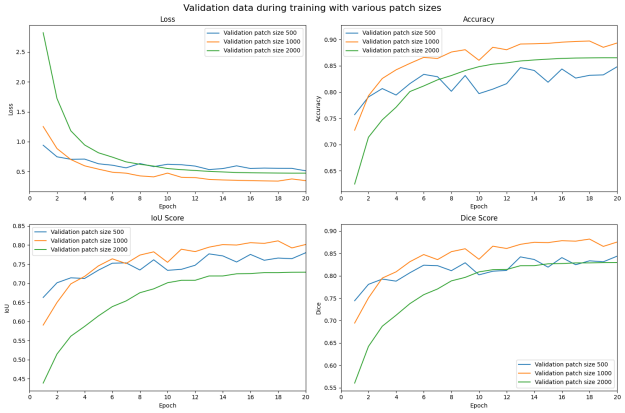

Figure 3: Validation history with different patch sizes, the architecture used is DeepLabv3+ Resnet34.

3.2 Comparison of Architectures

Table 5 compares the segmentation performance of our custom model and DeepLabV3+ with both ResNet34 and EfficientNet-B5 backbones, using the optimal patch size of 1000. Each network has been trained for 100 epochs, with early stopping if the validation loss did not improve after 14 epochs. Figure 4 illustrates the training progress of our custom model, starting with a low accuracy and IoU but gradually improving over the course of 100 epochs. Despite not being pre-trained and competing against state-of-the-art models, our custom model achieved in what seems to be an impressive 78% accuracy.

Architecture	Accuracy	mIoU	Dice	Loss
Resnet34	0.9114	0.8336	0.8969	0.2956
EfficentNet-B5	0.8969	0.8196	0.8859	0.3511
Custom	0.7841	0.6335	0.7329	0.7403

Table 5: Accuracy

Even if the accuracy seems high, as discussed previously, this metric alone does not tell the whole story. Table 6 reveals that our custom model struggled to generalize well on all the small objects such as humans, dogs, and fences. This limitation can be attributed to several factors. Firstly, the relatively small size and complexity of our model have hindered its ability to capture fine-grained details necessary for accurately segmenting small objects. Secondly, as evident from Figure 4, the model may have suffered from insufficient training, resulting in suboptimal performance. Consequently, while the model excelled at classifying large, homogeneous areas such as backgrounds, it faced challenges in precisely identifying and segmenting smaller entities within the images. Increasing the depth and complexity of the model, along with providing more diverse and comprehensive training data and a longer training in general, could potentially mitigate this limitation and improve the model's ability to

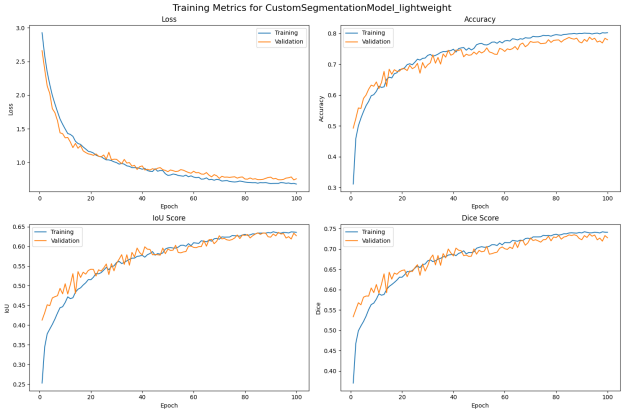

Figure 4: Custom architecture training history, the training took full advantage of all the 100 epochs, notice how it could have improved even more with more training, starting from a really high loss due to the lack of pre-training weights.

generalize across object scales.

Class	Resnet34	EfficentNet-B5	Custom
unlabeled	0.0009	0.0096	0.0072
paved-area	0.9351	0.9296	0.8018
dirt	0.5771	0.5118	0.3779
grass	0.9390	0.9068	0.8278
gravel	0.7932	0.7918	0.6767
water	0.9251	0.8409	0.6715
rocks	0.5395	0.4995	0.0000
pool	0.9390	0.8333	0.6859
vegetation	0.7004	0.6630	0.5219
roof	0.9012	0.8995	0.7240
wall	0.6446	0.6177	0.2502
window	0.5631	0.4892	0.0000
door	0.0000	0.0373	0.0000
fence	0.5300	0.4121	0.0000
fence-pole	0.0000	0.1392	0.0000
person	0.7134	0.7215	0.0029
dog	0.0000	0.0688	0.0000
car	0.9266	0.8162	0.0144
bicycle	0.5467	0.5565	0.0000
tree	0.6569	0.5906	0.1411
bald-tree	0.6265	0.4202	0.0000
ar-marker	0.6682	0.6626	0.0000
obstacle	0.6198	0.5924	0.1360
conflicting	0.0000	0.0000	0.0000

Table 6: Per-class IoU

On the other hand, despite the differences between ResNet34 and EfficientNet-B5, they achieved similar overall high accuracies (91.14% vs 89.69%) and mIoU (0.8336 vs 0.8196). However, a closer examination of the per-class IoU scores reveals nuanced strengths of each architecture. While ResNet34 exhibited consistent performance across most classes, EfficientNet-B5 demonstrated superior segmentation of small and challenging objects (see Table 6), achieving optimal consistency.
Specifically, ResNet struggled with categories such as fence-poles and dogs. Further analysis of our validation set revealed a significant class imbalance, with only a single instance of a dog present in the entire test set and a relatively low frequency of dogs in the training set as well. Consequently, the performance on the dog class is not representative of the models' overall capabilities but rather highlights a limitation of the dataset itself. A similar reasoning applies to the fence-pole class, albeit to a lesser extent. The models often misclassified fence-poles as regular fences due to their slim and elongated shape. Despite these challenges, ResNet34 demonstrated a generally balanced performance across the majority of classes.
On the other way, Efficientnet was able to identify these objects even if slightly, highlighting EfficientNet-B5's more complex architecture, which enabled it to capture fine-grained details and intricate patterns even in abscence of extensive data.
It is worth remembering that both the custom network and the pre-trained models faced challenges with batch sizes due to memory constraints, while the lighter architecture of ResNet34 allowed us to employ a larger batch size during training. This is a significant advantage, as previously explained. We believe this is the reason this architecture achieved overall higher accuracy albeit being less complex and less able to understand intricate patterns unlike EfficientNet-B5.
Overall, despite the differences between these two architectures and their training, they performed quite good since both models seems to understand the intricacies of the texture of the various classes, Figure 5 illustrates this strength, showcasing instances where the predicted vegetation masks surpass the accuracy of the ground truth labels, highlighting the models' ability to capture fine-grained local details. Furthermore, larger objects spanning multiple patches are correctly identified, demonstrating a robust understanding of global context.
Figure 7 shows ResNet34 training history, while Figure 8 shows EfficientNet-B5's.

Architecture	FPS 1000px	FPS 2000 px	FPS whole image	IoU 1000px	IoU 2000 px	IoU native image
Resnet34	29.57	87.20	135.53	0.8336	0.7910	0.6638
EfficentNet-B5	10.20	28.58	40.66	0.8196	0.7868	0.6603
Custom	16.93	55.35	108.54	0.6335	0.5828	0.4672

Table 7: FPS for each architecture with different patch size.

3.3 Real Time Performances

To evaluate the real-time performance of the architectures, we measured the frames per second (FPS) for each model. Leveraging the modular design of the dataset, we tested multiple test sets with different patch sizes, allowing us to trade accuracy for performance. The frame time was

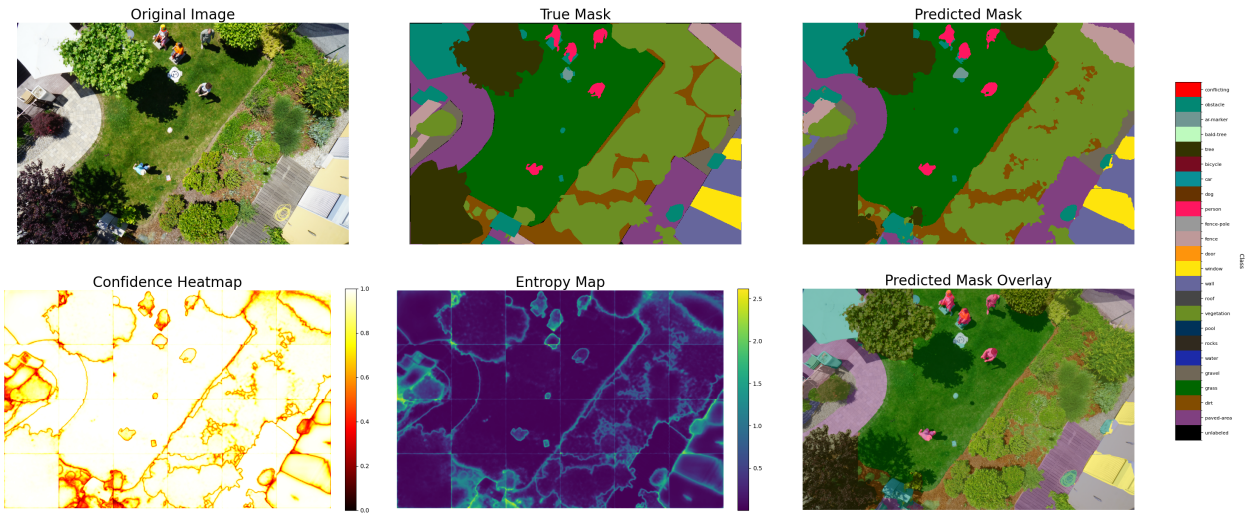

Figure 5: The results obtained using DeepLabV3+ with ResNet34. The confidence mask visualizes the value of the highest class probability after the softmax activation, indicating the model's confidence in the assigned class for each pixel. On the other hand, the entropy map illustrates the entropy of the class probabilities after the softmax activation for each pixel. Higher entropy values suggest that multiple classes have similar probabilities of being the final classification for a given pixel, indicating potential ambiguity or uncertainty in the model's predictions.

calculated as the sum of the inference time for each patch, the time to reconstruct the entire image, and the time to upscale the image to its original size. As shown in Table 7, the optimal balance appears to be achieved by using a patch size of 2000. This configuration resulted in an almost threefold increase in FPS compared to the baseline, albeit with a slight reduction in accuracy. This tradeoff demonstrates the flexibility of the patch-based approach, enabling the system to adapt to the specific requirements of real-time applications while maintaining an acceptable level of accuracy and still retaining rich information during training.

4 Conclusions

This study introduced a patch-based CNN approach with multi-scale processing for semantic segmentation of high-resolution aerial drone imagery. Key findings include:

- A patch size of 1000 pixels provides the best balance between detail and context.
- The multi-scale stitch level approach effectively captures features at various spatial resolutions.
- DeepLabV3+ with ResNet34 and EfficientNet-B5 achieve high accuracies, with ResNet34 exhibiting consistent performance and EfficientNet-B5 excelling at small objects.
- A patch size of 2000 pixels offers an optimal accuracy-speed tradeoff for real-time applications.

This study demonstrates the potential of patch-based CNNs for automated analysis of large-scale aerial datasets, paving the way for more effective environmental monitoring and land use mapping.

4.1 Future Work

This study lays a solid foundation for further research in semantic segmentation of high-resolution aerial imagery. Future work could explore the following avenues:

- **Expanded dataset**: Increasing the size and diversity of the training dataset could potentially improve the models' performance and generalization capabilities. A larger dataset would allow for more comprehensive training and evaluation, enabling the models to learn a wider range of features and adapt to various environmental conditions and help construct a more solid foundation for rare objects that lacked in this dataset.
- **High-performance computing**: Leveraging more powerful computational resources would enable experimentation with larger batch sizes and more complex models. This could lead to improved segmentation accuracy and the ability to capture finer details and intricate patterns in the imagery.
- **Domain adaptation**: Exploring domain adaptation techniques to bridge the gap between simulated or synthetic data and real-world aerial imagery could enhance the practicality and robustness of the segmentation models. This would involve developing

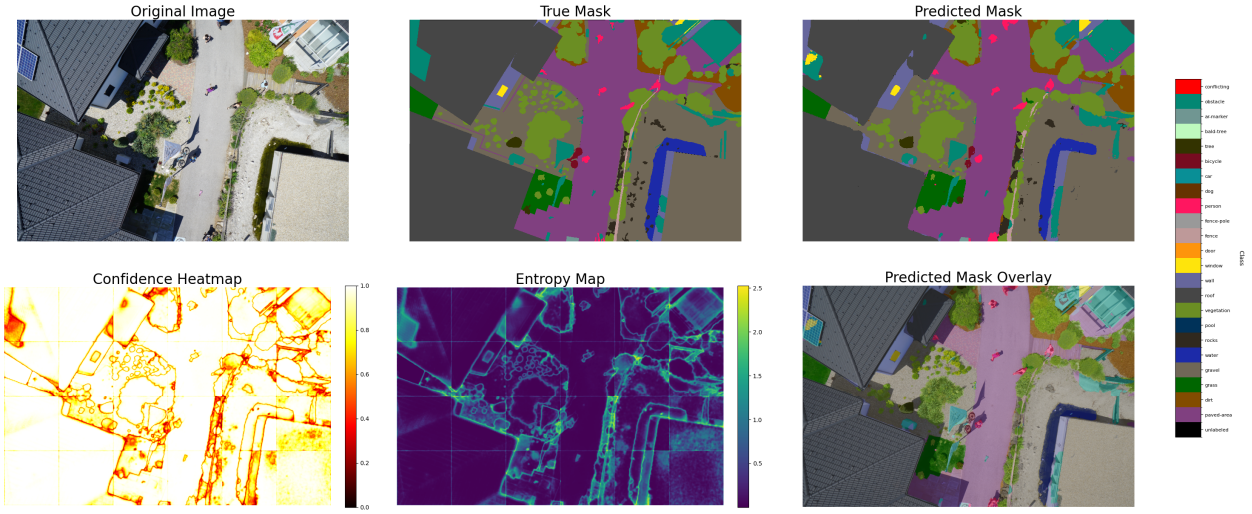

Figure 6: The results obtained using DeepLabV3+ with EfficientNet-B5.

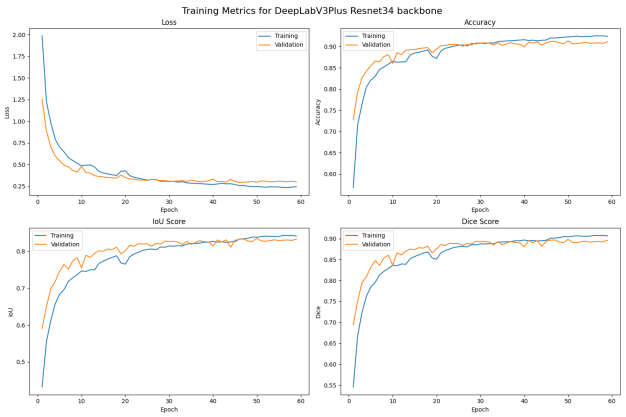

Figure 7: DeepLabV3+ Resnet34 training history, the training has stopped before reaching 100 epochs due to lack of validation loss improvement. Notice how the network converges without showing signs of overfitting.

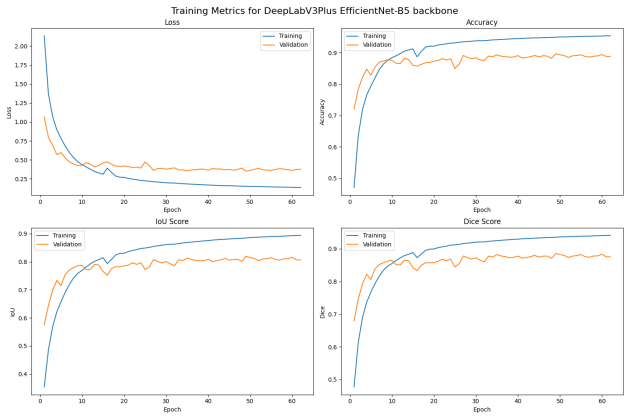

Figure 8: DeepLabV3+ EfficientNet-B5 training history, the training has stopped before reaching 100 epochs due to lack of validation loss improvement. Notice how the network converges less gracefully compared to the other architectures, even if without overfitting.

strategies to transfer knowledge learned from synthetic datasets to real-world scenarios, enabling more effective deployment in practical applications.

- **Active learning**: Investigating active learning approaches, where the models actively select the most informative samples for annotation, could reduce the manual labeling effort required for large-scale datasets. By prioritizing the most challenging or uncertain samples, active learning could optimize the annotation process and improve model performance with limited labeled data.
- **Real-time optimization**: Further optimizing the models and inference pipeline for real-time performance could enable their deployment in time-critical applications, such as autonomous navigation or interactive analysis. This may involve exploring model compression techniques and efficient architectures, enabling hardware accelerated drone to achieve high frame rates on the drone hardware itself.

Disclaimer

This report does not contain any form of plagiarism, including content generated or suggested by AI tools such as ChatGPT or similar services. All sources used have been properly cited and referenced.

REFERENCES

[1] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 40, pages 834–848. IEEE, 2017.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 833–851, 2018.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[4] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[6] Shuang Hu, Jin Liu, and Zhiwei Kang. Deeplabv3+/efficientnet hybrid network-based scene area judgment for the mars unmanned vehicle system. *Sensors*, 21(23), 2021.

[7] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[9] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[10] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 4898–4906, 2016.

[11] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[12] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision*, volume 115, pages 211–252. Springer, 2015.

[15] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[16] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[18] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[19] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016.