Real-Time Feature Extraction and Topological Change Detection of Multi-Biome Forests on Resource-Constrained Systems

ȚABREA LEONARD FABIAN, University of Twente

Abstract – Feature extraction and detection of changed topological characteristics of forest structures are essential in combating illegal logging and providing ecosystem parameters for environmental scientists. However, to this date, most models based on UAV[1] and satellite imagery, which have been developed in this regard, are not oriented towards real-time processing and inference. A key application of such lightweight models would be deploying them on a UAV to predict changes and extract features in real-time during flight. Therefore, this research paper aims to discover, develop and deploy semantic segmentation[2] machine learning models which are lightweight enough to run on resource-constrained systems, which could be mounted on a real UAV to perform these tasks. Towards this end, the following paper proposes 3 lightweight CNN[3] models trained on multi-biome forest datasets, capable of being deployed on two popular target microcontrollers. Moreover, a deployment pipeline for a drone capable of feature extraction and topological change detection is introduced as well.

Additional Key Words and Phrases: Machine Learning, Semantic Segmentation, Convolutional Neural Networks , Computer Vision, Resource-Constrained Systems, Image Processing.

1 INTRODUCTION

Illegal deforestation damages the ecosystem, increases soil degradation and has a net negative impact on local communities near rich woodland regions [20]. Moreover, according to the World Wide Fund for Nature [8], in 2002, approximately 70% of countries were affected by illegal deforestation, which highlights a global problem. As a result, extracting several important features of woodlands, such as canopy gaps, forest boundaries, and detecting area changes, remotely, remains of great interest to environmental scientists and specialists in combating illegal deforestation.

Furthermore, UAV imagery has been recently used in order to extract some of these features and detect canopy gaps and forest changes. [14] [20] [27]. Additionally, since satellite imagery is significantly more expensive than UAV imagery, there is also a growing interest in the deployment of on-site remote drones capable of collecting high-resolution data and images from an orthogonal top-down view in real-time [3] [28]. Unfortunately, currently, most UAVs act as remote-sensing systems and no real-time processing is done mid-flight. The data collected either has to be sent over the cloud to a more powerful server, which can make use of it to detect changes and patterns, or downloaded and processed when the UAV is returning to the docking station. Moreover, in remote and forest-dense regions such as the Amazon and the Congo Basin, cellular infrastructure lacks significantly [10], which makes the former alternative not even feasible. One could argue that satellite internet access could be used in such regions. However, to this date, such a solution for a small drone or UAV is more sophisticated, expensive, power-hungry and feasible only with small receivers which have low bandwidth and have been designed for telemetry and short data bursts, not secure HTTPS connections to cloud servers.

Nevertheless, for some specific applications such as combating illegal deforestation in remote areas, real-time inference and change detection in the structure of forests on the edge, mid-flight, remains largely unexplored and a real challenge. Addressing this issue could enable faster response times and damage minimisation [20]. All without relying on robust internet connectivity or power-hungry devices.

To this end, the following paper focuses on filling this gap by exploring, testing and compressing such models in order to be deployed on two target microcontrollers, which could be mounted on a UAV to perform edge inference, canopy gaps detection, and assess the expansion and/or contraction of forests over collected temporal data. Furthermore, to enhance the generalizability and expand potential use cases of this research, the models are trained on diverse multi-biome datasets. This is more demanding for the models in terms of accuracy since more global features have to be learned. However, the choice has been made in order to avoid picking an arbitrary biome and end up with a geographically biased model.

1.1 Problem Statement

Exploring, developing and deploying lightweight and accurate semantic segmentation models capable of detecting topological changes of multi-biome forests in real-time. Such models should also account for the limited space and computational resources of the two target microcontrollers.

1.2 Target Deployment Hardware

For the scope of this research, the models have been designed to accommodate the constraints and capabilities of two widely used, affordable microcontroller platforms, which are described in terms of their hardware resources in Table 1. However, in practice, these models are compatible with any C/C++ microcontroller which supports the TensorFlow Lite Micro Library [6] and its corresponding interpreter.

[1]UAV: Unmanned Aerial Vehicle: Aircraft flown autonomously or via remote control
[2]Semantic Segmentation: A technique which assigns a class label to each image pixel (e.g. forest vs. non-forest), which is well-suited to delineating canopy gaps or forest boundaries.
[3]CNN: Convolutional Neural Network

TScIT 43, 4 July, 2025, Enschede, The Netherlands

Table 1. Hardware resources of targeted microcontrollers

Microcontroller	Flash	RAM	LCFRB	Clk Freq
Arduino Nano 33 BLE	1MB	256KB	200KB	64MHz
ESP32 WROOM	4MB	520KB	107KB	160MHz

Besides the limited computing power, RAM and FLASH memory, another key limiting factor when it comes to deploying Edge ML Models on microcontrollers is the LCFRB[4] available for allocation during compile time for a custom program. Note that even though the ESP32 has more RAM space in comparison to the Arduino, according to Table 1, the LCFRB (≈ 100KB) is significantly smaller since its memory is fragmented by hardware design. This will impose an upper bound for the size of the tensor arena[5] which can be allocated to the interpreter at runtime. Note that this is the case for any Edge ML interpreter since it will have to sequentially load layers and perform intermediate operations in the tensor arena. Therefore, layers with many parameters which require more intermediate storage space for calculations cannot be used, e.g. a concatenation of two tensors of size 64x64x16 is not feasible, nor a dense layer with more than 1024 neurons or just a simple convolution applied to a tensor with many channels.

1.3 Model Constraints

In order to make the models deployable on the target hardware, namely the Arduino Nano 33 and the ESP32, the following constraints have been imposed throughout the research and development phase. The tensor arena, model size and parameter count are hard constrains since otherwise the model cannot be physically deployed. The others are quality constraints, such as performing inference in a reasonable time frame.

- Model Size: < 1000Kb (Without Quantization - Float32)
- Total Parameters: < 500K
- Tensor Arena Size: < 100Kb
- Total FLOPS: < 100M
- Inference Time (64×64×3 Input Tensor): < 3s

1.4 Research Questions

RQ1: Which lightweight CNN models perform the best when it comes to binary semantic segmentation of UAV multi-biome forest imagery across space and time complexity, and what is the trade-off of each such model?

RQ2: How to detect topological changes and extract features of forest structures on a resource-constrained device, mounted on a UAV in real-time?

RQ3: Which techniques and architectures are the best to reduce the size of the CNN models enough to be deployed on a resource-constrained device while conserving the model accuracy as much as possible?

[4]LCFRB: Largest Continuous Free Ram Block
[5]Tensor Arena: Static RAM buffer for inputs, outputs, and intermediate tensors used by the interpreter during inference.

2 RELATED WORK

Important contributions made by Jiahong et al. [28] in detecting forest patches and forest coverage using CNN-like models on UAV imagery have been done in late 2024. The results showed a 0.98% accuracy and a precision of 0.91 using a UAV flying at a distance of 8 meters. However, it is important to note that these models were very large relative to the memory resources of a microcontroller and not intended to run on resource-constrained devices, even though the data acquisition was made with UAVs.

Similar work has been done by Htun et al. [14] in detecting canopy gaps in uneven-aged mixed forests in Japan using the U-Net and the ResU-Net models. The research concludes that the ResU-Net was the superior model, achieving an accuracy of 0.96 on UAV imagery datasets. However, similar to the previous contribution, the models were not intended to run on resource-constrained devices. Moreover, the data included additional dimensions from LiDAR sensors and Trichromatic RGB cameras.

When it comes to general techniques researched for model compression, such as object detection models and CNNs, Mahmoudi et al., Alexandre et al., and Sambhav et al. [18], [17], [15] explored key methods such as 8-bit quantization, knowledge distillation, and block pruning.

2.1 Commentary on Related Work

The aforementioned models, developed by Jiahong et al. [28] and Htun et al. [14], have architectures which are unfeasible to be deployed on resource-constrained devices. For example, even a standard UNET [21] cannot be deployed on a microcontroller with its default number of filters, concatenation layers and upsampling layers. Moreover, using a backbone such as the ResNet101[11] used by Htun et al. [14] to augment the models is not possible since such a backbone has ≈ 44.5M parameters [7]. Therefore, the problem statement of this paper remains largely open.

Furthermore, the dataset used for training by Jiahong et al.[28] is relatively small (only 2000 images of size 512x512 px) while being sampled from only one region: Tieliugang, Sanya City, Hainan Province, China. Thus, it is not a multi-biome dataset. Thus, there is no burden of geographical generalisation for the models. For example, identifying trees in tropical, temperate, mountain and desert regions at the same time becomes a significantly harder task..

3 METHODOLOGY

3.1 Performance Evaluation

The following formulas were used to compute the accuracy, F1 scores, precision, recall, and IoU values. Note that they are applied pixel-wise since the predictions are made for every individual pixel.

$$\text{Intersection over Union (IoU)} = \frac{TP}{TP + FP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.2 Data Acquisition

During the data acquisition stage, the OAM-TCD [26] Dataset of size 4,169 and the VHRTrees [24] of size 10,674 have been discovered and inspected to train the models. Under a quality inspection, the OAM-TCD has better mask labels, which have a higher granularity and pixel accuracy, since a significant part of the dataset has been annotated manually. Moreover, the OAM-TCD has a higher resolution and is a more diverse dataset, if not the most diverse dataset of forest imagery spread across multiple terrestrial biomes, according to its authors, who compiled, labelled and curated the dataset in 2024 [26]. All 14 covered terrestrial biomes and their distribution across the dataset can be observed in Figure 1 below.

Cross-validation fold	1	2	3	4	5	Holdout	Total
Biome							
Unmatched	46	18	34	39	20	59	206
(1) Tropical and subtropical moist broadleaf forests	268	412	378	377	295	276	2006
(2) Tropical and subtropical dry broadleaf forests	53	68	12	48	46	13	240
(3) Tropical and subtropical coniferous forests	11	6	13	19	10	6	65
(4) Temperate broadleaf and mixed forests	354	368	260	371	277	202	1832
(5) Temperate coniferous forests	74	43	51	46	28	55	297
(6) Boreal forests/taiga	0	0	0	8	0	3	11
(7) Tropical and subtropical grasslands, savannas, and shrublands	15	34	20	28	37	8	142
(8) Temperate grasslands, savannas, and shrublands	4	5	14	15	12	34	84
(9) Flooded grasslands and savannas	0	10	0	0	0	0	10
(10) Montane grasslands and shrublands	0	6	12	0	0	0	18
(11) Tundra	0	9	0	0	0	0	9
(12) Mediterranean forests, woodlands, and scrub	12	11	27	16	12	2	80
(13) Deserts and xeric shrublands	1	13	6	1	6	7	34
(14) Mangrove	12	0	2	1	12	1	28
Total image tiles	850	1003	829	969	755	666	5072

Table 1: Number of image tiles in OAM-TCD for each terrestrial biome class, for each cross-validation fold in the dataset. "Unmatched" tiles were unable to be matched to a biome via polygon intersection. Also shown is the number of tiles per fold, and the distribution of biomes throughout the entire dataset.

Fig. 1. Distribution of images in terrestrial biomes, and in each of the suggested cross-validation folds. Source: The OAM-TCD Paper [26]

The OAM-TCD dataset contains images of forests from an orthogonal view of size 2048x2048 px with a resolution of 10 cm per pixel. Since feeding such an input image and/or parts of it to a model deployed on a microcontroller is not feasible, the images have been downscaled by a constant of 2^3 in both height and width, leading to the final dataset with images of size 256x256 px. While it is possible to compress the images even further, this will reduce the final pixel resolution on the ground and will provide less spatial context for the models.

Therefore, by keeping the images of size 256x256 px, each compressed pixel will correspond to 80 cm on the ground. Thus, each square pixel will cover an area of $80 * 80 = 6400$ cm^2 $= 0.64$ m^2, which is roughly the basal area of a mature tree. Therefore, an inherent trade-off has already been encountered due to pixel interpolation during the compression stage, which hinders the contribution of sparse tree regions and edges. However, the areas which cover relatively dense spots of pixels, either corresponding to trees or background, are conserved. For example, if for a block of 8x8 px in the original image more than 32 pixels are mapped to trees on the ground, the compressed representative pixel will be mapped to a tree as well. In the end, all 256x256 px images have been further partitioned into 64x64 px sub-images since this will be the spatial dimension of the input tensor fed into the models. The reason why this input dimension has been chosen it's because it is the largest size tested in a sequence of powers of two, which fits in the LCFRB of the ESP32. Thus, the raw dataset consists of 66,704 (64x64 px) RGB images.

3.3 Data Augmentation and Class Balancing

Since the dataset had a pixel-level imbalance, i.e. the overall number of pixels corresponding to the background was larger, the dataset has been balanced and augmented using a minority oversampling technique, trying different augmentation methods. In essence, a cyclic random traversing algorithm randomly picks images from the dataset, and if the number of white pixels of the mask (label) which corresponds to the RGB image is at least 50% higher than the number of black pixels in the same image, a synthetic augmentation of the image is added to the dataset. This process is repeated until the total ratio of white to black pixels becomes 1. In the end, the final balanced augmented dataset has a size of 114,272, which means that it has been synthetically expanded by 71.31%.

One of the augmentation methods used was the random grid shuffling technique [12] for forest datasets as described by Josh et al. In essence, the corresponding image-mask pair have been divided using an 8x8 grid, and using the same random seed, the tiles were rearranged for the augmented image-mask pair result. Such a depiction can be seen in Figure 2 below.

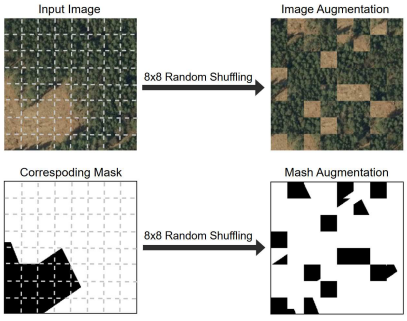

Fig. 2. Data Augmentation using the random grid shuffling technique

With such a technique, an improvement of $\approx 2 - 3\%$ in the accuracy of the training dataset has been observed. However, in comparison

to using random rotations, random horizontal flips, and random vertical flips, the accuracy for the validation dataset has been reduced by $\approx 1-2\%$, which means that the model started to learn unnatural patterns in the data. Therefore, only the second augmentation method has been used in the end.

3.4 HyperParameters and Model Training

Since the augmented dataset is pixel-wise balanced, the common loss function for all the models is the binary cross-entropy[6] function. The reason is that both predicted labels are considered to have the same weight. In other words, it is assumed that overpredicting FN[7]s (black pixels - false indication of forest loss) is as bad as overpredicting FP[8]s (white pixels - false indication of forest grain).

During training, the Adam Optimiser with a learning rate of 0.001 was used. Additionally, the batch size has been chosen to be either 4 or 32, as it was found to provide better granularity and feature extraction during training in comparison to other batch sizes in powers of two. Both of these parameters have been optimally chosen after performing a grid search targeting the maximisation of the validation accuracy after 5 epochs of training. To split the dataset, a standard 80%, 10%, 10% train, validation, test split has been employed for partitioning since the entire dataset is sufficiently large (>100K) data points, to allow for such a split, which allocates more data points by percentage for training purposes. In the end, the testing set is of size 11,428 images, which have been randomly picked from the balanced dataset. Additionally, the validation dataset of 11,828 is used in conjunction to evaluate the confidence of accurate predictions on unseen data.

The output activation function of each model is sigmoid[9] since the output tensor of each model has only 1 channel. This is the greyscale confidence probability map with values ranging from 0 to 1. Thus, to consider a pixel as positive or negative in the final labelling, a threshold must be applied. This threshold has been adjusted dynamically in order to maximise the accuracy for each model after training using the testing set. The step used for adjusting the threshold iteratively is 2.5. The best values range from 40% to 60%. This process can be referred to in Appendix A1. Moreover, visualisations of predicted masks can be observed in Appendix B.

3.5 Development Workflow

The models have been developed, trained and validated using the Python Keras API from TensorFlow in Jupyter Notebooks. The environment used for development was the JupyterLab Platform hosted by UTwente. In this way, the models have been trained using a much more powerful GPU in the cloud - the NVIDIA 8-core A10 with compute capability 8.6 and cuDNN version 90300.

After each model has been trained and validated, it has been compressed into a .tfml file. From the .tfml file, a Python script converted the model into two .cc and .h files. These C files are publicly available and ready to be used for actual deployment. Furthermore, the models have been tested and run natively on the Arduino Nano 33 and ESP32. Using the Arduino_TensorFlowLite and the Chirale_TensorFlowLite libraries for the Arduino and ESP32, respectively. For inference, an image which was converted to a C Array as a constant expression has been encoded in the FLASH. When burning and running the sketch on the microcontroller, the image is fed to the interpreter and the inference using the model's weights is done in a continous loop. For each inference, the time that it takes to get a full prediction is measured using the built-in micros() function of the microcontrollers. Note that it is enough to store only one encoded image of size 64x64 px on 3 RGB channels to measure the inference time since the performance is deterministic in this stage and independent of the actual values of each pixel. This process can be referred to in Figure 3.

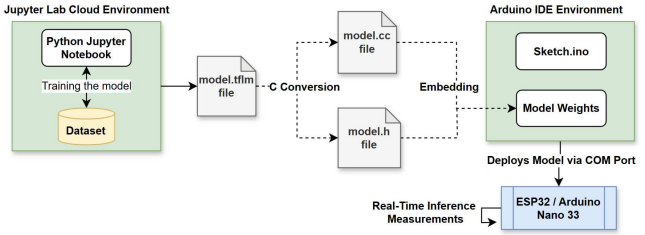

Fig. 3. Development Workflow - Training and Native Deployment Process

4 PROPOSED MODELS

During the research phase, three main models which showed promising results have been developed. All three models are based on well-known architectures which have been progressively altered and cropped, in order to keep most of their initial accuracy while drastically decreasing their size such that they can be deployed on the targeted microcontrollers, adhering to the memory limitations.

4.1 Tiny UNET with a Stacked Depthwise Convolutional Block

This model is based on the standard UNET[21] architecture, which has been augmented with a Stacked Depthwise Convolutional Block before the output layer. Moreover, the model uses Add operations instead of Concatenate operations to fit into the target tensor arena. The activation functions for the convolutional blocks have also been replaced with Hard Swish [13]. This decision was made based on the insights gained from the research done by Avanash et al. [1], who proved that this activation function leads to better performance in the case of semantic segmentation of satellite images instead of using ReLU or ReLU6. This change accounted to an average increase in accuracy of $\approx 1.5\%$. Moreover, a Stacked Depthwise Convolutionl Block of 3 layers with decreasing kernel sizes (3x3, 2x2, 1x1) in this order and ReLu activation functions have been introduced at the end to act as a feature refinement. The reason for this is the fact that the compressed version of the UNET lost prediction accuracy due to replacing the concatenation layers with additive layers for the

[6] BCE is an loss function defined as: $\mathrm{BCE}(y, \hat{y}) = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$
[7] FN: Incorrectly labelling a positive data point as negative
[8] FP: Incorrectly labelling a negative data point as positive
[9] Sigmoid is an activation function defined as: $sigma(x) = \frac{1}{1+e^{-x}}$

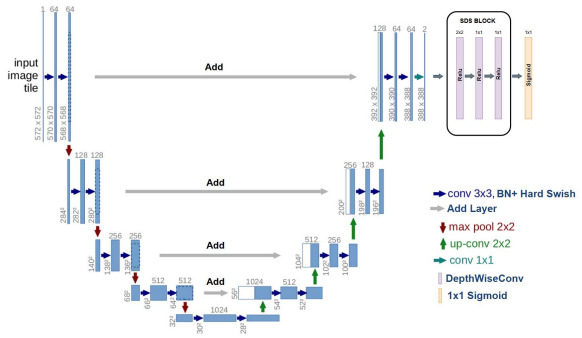

Fig. 4. Tiny UNET with a Stacked Depthwise Convolutional Block. Adapted from: www.lmb.informatik.uni-freiburg.de/people/ronneber/u-ne [22]

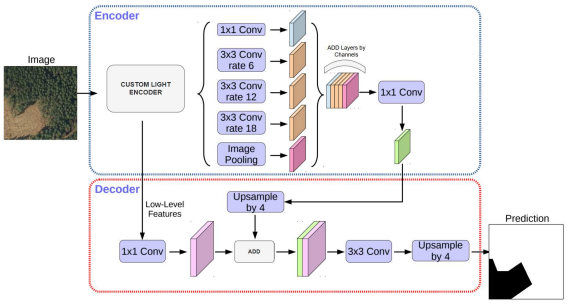

Fig. 5. Tiny DeepLabV3+ with Light Encoder. Adapted from: www.sh-tsang.medium.com [25]

skip connections. Thus, this measure aims to offset this loss. The introduced block adds minimal space and time complexity by adding only 157 new parameters while increasing the base model's accuracy by ≈ 0.95%. The number of filters has also been adjusted to range from 3 up to 24 in the encoder and decoder. The reason is that having more than 3 filters during the first convolution when the tensor has its full spatial dimension (64x64) will not fit into the tensor arena. Thus, a characteristic sequence of $(2+1, 4+2, 8+4, 16+8, ..., n+\frac{n}{2})$ has been used to adjust the number of filters for each double layer in the encoder. The reserve of this sequence is used for the decoder.

4.2 Tiny DeepLabV3+ with Lightweight Encoder

This model is based on the architecture of the DeepLabV3+[5] introduced by Google. The original architecture was altered by adding a lightweight initial Encoder instead of the original Atrous Convolutional Encoder, which was too heavy. This encoder is made up of only 4 convolutional layers linked in pairs. Each pair having 24 and 58 filters, respectively. The Output Stride, which is 4 in the original model, has been set to 2. The reason is that the input tensors for all the models of this paper have a very small spatial dimension of only 64x64. Using the original Output Stride will pass a tensor of size only 4x4 to the Atrous Spatial Pyramid Pooling ASPP[10], which is too small to benefit from it in the first place. Thus, in the current implementation, the tensor fed into the ASPP (The special block for DeepLabV3+) is 8x8, which enhances the spacial context and the features learned. This improved the model's performance by ≈ 2.5%. Moreover, the Concatenate operation has also been replaced by the Add operation as it was previously explained why this is necessary. To account for this loss of context across channels, a double decoder layer with 48 filters has been introduced at the end, but only with a stride of 1 to avoid distorting the spatial dimension of the tensor before the final upsampling layer which has also been adjusted to upsample by only 2 before the last output layer.

[10] Atrous Spatial Pyramid Pooling Block: Block which uses several parallel atrous convolutions at multiple dilation rates, all concatenated at the end for context inference.

4.3 Tiny SegNetV2 with Dense Layer Bottleneck

This model is mainly built on the original SegNet Architecture [2] introduced by Vijay et al. in 2015. The main difference, however, is that two fully connected dense layers of 64 neurons each have been introduced in the bottleneck. This technique proved to improve some of the binary semantic segmentation tasks as it was discovered by Brahmbhatt et al. in 2019 [4]. This variation has been done in an effort to account for the fact that the pooling indices operations have been removed. This addition increases the base model's accuracy by 2.32%. Moreover, the number of filters has also been adjusted using the same characteristic sequence presented for the Tiny Unet.

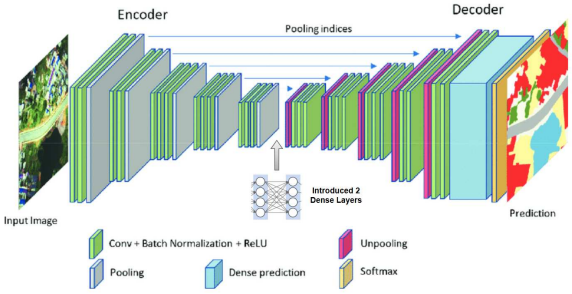

Fig. 6. Tiny SegNet with Dense Layer Bottleneck. Adapted from: The original 2015 SegNet paper [2]

4.3.1 Notable Metions. Two standard Unet Architectures have been tested as well for completeness during the exploratory phase. Namely, the Attention Unet introduced by Oktay et al. [19] in 2018 and the Unet++ introduced by Zongwei et al. [29] in the same year. Unfortunately, both architectures yielded modest results. Thus, no effort was made to extend them. The architectures have been deployed using a classic hour-glass pattern of 6 layers, made up of (2,4,8,16,32,64) filters and ReLu activation functions for each layer in the encoder and the same order of filters, but reversed in the decoder. Both models yielded an accuracy between 84% to 87%.

5 PROPOSED DEPLOYMENT PIPELINE

The Deployment Pipeline can be captured completely in 5 stages, which are also depicted in Figure 8.

Prerequisites: Before deploying the drone with the mounted microcontroller and camera, the region of interest should first be tessellated into square cells. A tessellation is a division of a surface into adjoining shapes (tiles) that completely cover the area with no gaps or overlaps. This process is described by Griffith et al.[9] in their paper on this subject. In practice, this can be done by using already-made software such as the DGGS [16].

Moreover, a GPS module such as the u-blox ZED-F9P compatible with the Arduino 33 Nano and the ESP32 should be mounted on the drone in order to asses the geographical tile over which is hovering during the fly. For ease of use, the drone could fly in a deterministic path. In the end, it should be noted that in reality, there are some limitations and errors to be considered if this tiling process is used due to the Earth's curvature. However, most drones and UAVs fly at a relatively low altitude, and these errors for this scope can be largely ignored.

Stage 1: During the first stage, an RGB image over a particular geographical tile is captured by the drone's camera during flight.

Stage 2: The second stage corresponds to partitioning the original image into (D x D) smaller sub-images and sequentially feeding then as input tensors with three RGB channels into the model. For the scope of this research, the D parameter is 64 since it is the largest power of two which fits in the available LCFRB of the ESP32 as an input tensor.

Stage 3: The third stage is the inference stage, which takes the input tensor, computes all the intermediate convolutions and outputs a tensor with the same spatial dimension but with only 1 channel, which are confidence values for each pixel ranging from 0 to 1 (the greyscale probability map).

Stage 4: In the fourth stage, the image is thresholded using the best-performing threshold, which was dynamically discovered after the training of each model. However, this is static during deployment. Thus, each pixel will have a binary value in the end. Moreover, for the same snapshot (in case this is not the first snapshot for the corresponding geographical tile), there should exist in the memory of the microcontroller at least one most recent snapshot (saved only as a binary mask in the flash memory of the microcontroller). The image differencing function in this stage will detect topological changes with a double-pass linear complexity in terms of the stored spatial dimension (D x D).

Stage 5: Topological change detection — such as forest loss or gain and gap detection, are identified at this stage remotely by the microcontroller using the simple image differencing technique previously described. Note that for this, it is enough to store only the binary masks since they occupy less space due to the fact that they have only 1 channel and can be efficiently compressed. The way how these characteristics are determined is detailed in the Post-Processing section at the end.

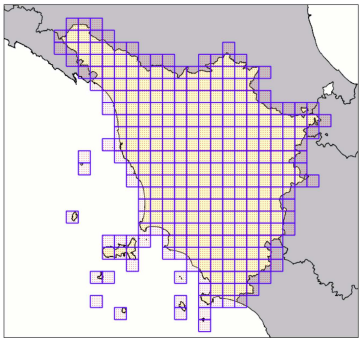

Fig. 7. Example of geographical tesselation covering Tuscany with square tiles of size 10x10 Km. Source: www.gaia-gis.it [23]

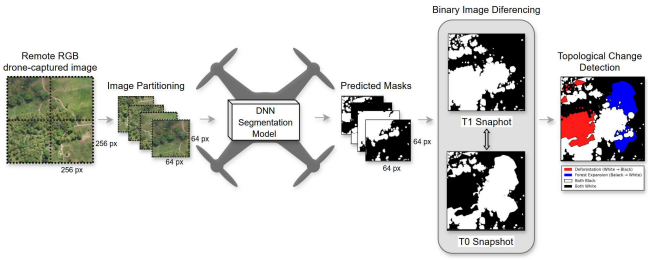

Fig. 8. Deployment Pipeline in 5 stages

6 DATA ACQUISITION, AUGMENTATION AND BALANCING

7 OPTIMIZATIONS

7.1 8-Bit Quantization

In order to compress the models and keep them deployable while adhering to the Tensor Arena Size constraint, a uniform affine 8-bit quantization has been applied as it was introduced by Sambhav et al. [15] in 2020.

$$\text{Quantized: } q = \text{round}\left(\frac{r}{s}\right) + Z \quad \Longrightarrow \quad r \approx (q - Z) \times s$$

where:

- r: original real (floating-point) value
- s: scale factor ($s = \frac{\text{float}_{\max} - \text{float}_{\min}}{\text{int}_{\max} - \text{int}_{\min}}$)
- Z: zero-point (maps $r = 0$ to the integer zero-value, ensuring exact representability)
- q: quantized integer in the 8-bit range $[\text{int}_{\min}, \text{int}_{\max}]$
- round$(\cdot)$: nearest-integer rounding operation

Performance drop after quantization: During the quantization stage, an average performance drop of no more than ≈ 2% for accuracy, ≈ 2.5% for the F1 score, ≈ 3% for IoU and ≈ 3% for precision and recall has been recorded for each model. This still makes the models feasible for deployment without a significant drop. The overall performance of the best model in terms of accuracy (DeepLabV3+) is decreased by only ≈ 1.5%, which is reasonable considering that all the weights have been trimmed from Float32[11] values to UInt8[12] values. All models have been successfully quantized and deployed on the target microcontrollers, since without it, the deployment is not possible.

7.2 Manual Micro Operator Selection

Manually configuring the operators required by the model when configuring the TensorFlow Lite Micro (TFLM) interpreter greatly reduced the tensor arena requested size. This approach replaces the use of a default interpreter (such as AllOpsResolver) that loads support for all available operations, regardless of whether they are needed. By limiting the interpreter to only the necessary operators for each model during loading, the memory footprint of the tensor arena used by the interpreter during inference is significantly reduced.

7.3 Overfitting Mitigation

Furthermore, when it comes to avoiding overfitting, the following measures have been taken:

- Random batch shuffling during each epoch
- Early Stopping with a patience of 10 epochs and tolerance value for the validation accuracy of 0.001
- Introduced 4 to 8 uniformly distributed dropout layers in each architecture with a drop probability of 20% as a regularisation technique

8 RESULTS - PERFORMANCE ANALYSIS

8.1 Model Performance - The winner for each category is highlighted in light green

When it comes to accuracy and overall performance, the Tiny DeepLabV3+ with the Lightweight Encoder performs the best when it comes to all evaluation metrics. On the second spot comes the Tiny Unet with the SDC Block, with an accuracy drop of only 2.5% and a more significant F1 Score drop of 3.61%. Unfortunately, the Tiny SegNetV2 underperformed in every category. These comparisons can be directly observed in Table 2 below, while the learning curves can be referenced in Appendix C .

Table 2. Performance over the testing dataset after training the models and adjusting the best-performing threshold for each of them.

Model	Loss	Acc	F1	IoU	Recall	Prec
Tiny DeepLabV3+	0.2299	0.9085	0.8817	0.8074	0.8888	0.8907
Tiny UNet	0.2977	0.8835	0.8456	0.7624	0.8581	0.8625
Tiny SegNetV2	0.3694	0.8537	0.8064	0.7098	0.8127	0.8399

[11] Float32: 32-bit floating-point format for precise real numbers in ML models.
[12] UInt8: is an 8-bit unsigned integer data type ranging from 0 to 255.

8.2 Native Model Space and Time Complexity

When it comes to time complexity, the Tiny Unet wins when it comes to inference time, as it can be seen in Tables 4 and 5. While the difference on the Arduino Platform is marginal, on the ESP32, it is two times faster in terms of inference time. Furthermore, the Tiny Unet is 5.03 times smaller when it comes to FLASH Memory Size, which makes it fit for memory-constrained microcontrollers.

Table 3. Comparison of Models Native Size and Theoretical Computational Requirements

Model	Parameters	Memory Size	Total FLOPs
Tiny U-NET	49,223	192.28KB	10,344,854
Tiny DeepLabV3+	247,993	968.72KB	98,061,088
Tiny SegNetV2	104,562	408.45KB	13,777,646

Last but not least, the tensor arena allocated for each model has been measured as well. This is the reserved RAM space during runtime for the interpreter to load the most computationally intensive intermediate layer. This is a crucial aspect for deploying models on microcontrollers since most of them have limited allocatable RAM space during runtime, and using a model which does not require too much RAM is necessary if other tasks and jobs have to be done on the microcontroller as well, besides just running the model. In this department, the Tiny SegNetV2 wins by a small margin of 4KB and 7KB relative to the Tiny Unet and the Tiny DeepLabV3+, respectively. However, the Tiny SegNetV2 has the largest inference time on both platforms, while being the least accurate model. In consequence, this makes the memory gain obsolete.

Table 4. ESP32 Inference Performance by Model: Inference Time and Tensor Arena Size

Model	Inference Time	Tensor Arena Size
Tiny UNET	0.87s	86KB
Tiny DeepLabV3+	1.64s	89KB
Tiny SegNetV2	1.852s	82KB

Table 5. Arduino Nano 33 Inference Performance by Model: Inference Time and Tensor Arena Size

Model	Inference Time	Tensor Arena Size
Tiny UNET	2.56s	87KB
Tiny DeepLabV3+	2.68s	90KB
Tiny SegNetV2	5.91s	83KB

8.3 Comparative Analysis - Benchmarking

Unfortunately, a direct comparative analysis on the OAM-TCD Dataset with lightweight models is not possible since no public models lightweight enough to fit on a microcontroller are known as of June 2025. However, the authors published benchmarks of large generic semantic segmentation models as they can be seen in the table below. However, it should be noted that these models

have been trained using random crops of 1024x1024 px instead of 64x64 to provide more spatial context. Under a simple inspection, it seems like the proposed Tiny DeepLabV3+ Model exceeds most of the authors' models in terms of Accuracy and F1 Scores while being much lighter and having less spatial context. However, as previously stated, since the original images are compressed to 256x256 px and sub-images of 64x64 px are extracted, fine-grained details are lost due to interpolation. Therefore, the models have to some extent an easier and more coarse task to accomplish in comparison to the models introduced by the authors of OAM-TCD. However, it should be noted that even though the task is more coarse, the proposed models in this paper have to work with a spatial context two times smaller than the one used by the authors (16 sub-images instead of 4). This increases the difficulty of correct pixel classification.

Table 6. Semantic segmentation model performance on the OAM-TCD holdout set [26]. These baseline models have been tested and benchmarked by the authors on the dataset OAM-TCD.

Model	IoU	Accuracy	F1	Parameters
UNet ResNet34	0.838	0.883	0.871	24.4M
UNet ResNet50	0.849	0.881	0.880	35.5M
SegFormermit-b0	0.865	0.892	0.882	3.72M
SegFormermit-b1	0.870	0.897	0.891	13.7M
SegFormermit-b2	0.871	0.889	0.898	27.3M
SegFormermit-b3	0.875	0.884	0.875	47.2M
SegFormermit-b4	0.875	0.891	0.901	64M
SegFormermit-b5	0.876	0.890	0.902	84.6M

9 POST PROCESSING

9.1 Topological Change Detection

The final prediction is a binary mask (white pixel = tree, black pixel = background) delineating forest cover for each drone-captured snapshot. Topological change detection is performed through simple image differencing of consecutive masks. If a pixel transitions from white to black, it indicates forest loss in that region; conversely, a black-to-white transition signals gain. This per-pixel comparison naturally highlights areas of deforestation or reforestation.

In order to extract correct topological features, it is assumed that the UAV Vehicle that is capturing the images is flying consistently at the same altitude. In this fashion, depending on the aperture and resolution of the camera, it is possible to compute a direct mapping between a digital pixel to an (N × N) square meter tile on the ground, i.e. finding the space resolution of the image.

9.2 Topological Features Extraction

Once the binary masks are generated, several topological and geometric metrics can be computed using only the binary predicted values:

- **Forest sparsity**: the ratio of black to white pixels, reflecting canopy fragmentation.
- **Centroid**: the geometric centre of white regions, useful for tracking forest patch displacement or expansion.

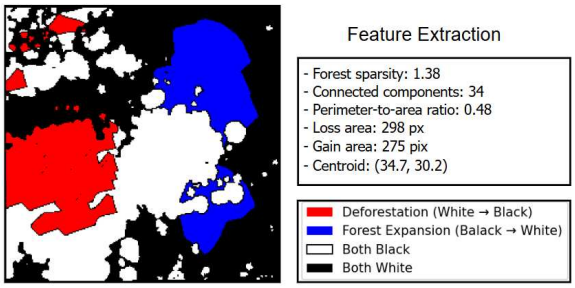

Fig. 9. Example of feature extraction after applying image differencing over two temporal images covering the same geographical tile.

- **Connected components**: counts of discrete forest patches and their individual areas.
- **Perimeter-to-area ratio**: a shape descriptor indicating edge complexity (e.g. compact vs. irregular forest patches).
- **Change area**: total count of newly black or white pixels, quantifying net loss or gain.

10 CONCLUSION

The Tiny DeepLabV3+ performed the best in terms of inference time on all platforms. However, it is the heaviest model (968.72 KB). For a small performance drop of ≈ 2-3% across all metrics, if significant FLASH memory limitations exist during deployment, the Tiny Unet can be used with much confidence as well since it still maintains most of the accuracy of Tiny DeepLabV3+ while being five times smaller (192.28KB) and faster in terms of inference time on both platforms. In this case, it could be argued that the Tiny Unet delivers more performance per parameter count, and it is the most efficient model if equal weights are put on time and space efficiency. The Tiny SegNetV2 model can be discarded since it doesn't outperform the other two models in any department, except for a negligible advantage in the tensor arena, while severely lacking in terms of space and computational efficiency.

In the end, the results are overall satisfying when it comes to the performance of the proposed models relative to their tiny size and the other models' benchmarks provided by the authors of OAM-TCD[26]. However, a direct comparison still cannot be done due to different input tensor spatial dimensions. Moreover, all default implementations of the original architectures of the proposed models use over 1M total parameters, which have been greatly reduced without sacrificing too much performance for the proposed Tiny ML variations. The models also have a good F1-score, balancing the False Positives and False Negatives well.

11 PRACTICAL REMARKS

In the public GitLab repository of this project: (https://gitlab.utwente.nl/s2995735/reserachprojectforestsegmentation), all results can be reproduced and all model weights are provided in standard formats: .weights.h5 (trained weights), .tflm (quantized TFLite Micro model), and the correspoding conversions to C source/header files (model.cc, model.h) for embedded deployment on C/C++ microcontrollers.

REFERENCES

[1] R. Avenash and Prashanth Viswanath. 2019. Semantic Segmentation of Satellite Images using a Modified CNN with Hard-Swish Activation Function. In *VISIGRAPP*. https://api.semanticscholar.org/CorpusID:88483500

[2] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. 2015. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling. arXiv:1505.07293 [cs.CV] https://arxiv.org/abs/1505.07293

[3] Tiberiu Paul Banu, Gheorghe Florian Borlea, and Constantin Banu. 2016. The use of drones in forestry. *Journal of Environmental Science and Engineering B* 5, 11 (2016), 557–562.

[4] P. Brahmbhatt. 2019. Skin Lesion Segmentation using SegNet with Binary Cross-Entropy. In *Proceedings of the International Conference on Artificial Intelligence and Speech Technology (AIST2019)*. India, 14–15.

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*. 801–818.

[6] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. 2021. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. In *Proceedings of the 4th MLSys Conference*. https://arxiv.org/abs/2010.08678 arXiv:2010.08678.

[7] Vijaypal Singh Dhaka, Sangeeta Vaibhav Meena, Geeta Rani, Deepak Sinwar, Kavita, Muhammad Fazal Ijaz, and Marcin Woźniak. 2021. A Survey of Deep Convolutional Neural Networks Applied for Prediction of Plant Leaf Diseases. *Sensors* 21, 14 (July 2021), 4749. https://doi.org/10.3390/s21144749

[8] World Wide Fund for Nature. 2002. *Forests for Life: Working to Protect, Manage & Restore the World's Forests*. WWF.

[9] Daniel A Griffith, Crystal Conway, Americo Gamarra, Mae Hutchison, Dongeun Kim, and Yalin Yang. 2025. Political Districts Versus Customized Polygons: Implementing Geographic Tessellation Stratified Random Sampling. *Transactions in GIS* 29, 2 (2025), e70019.

[10] Leon Gwaka, Müge Haseki, and Christopher S Yoo. 2023. Community networks as models to address connectivity gaps in underserved communities. *Information Development* 39, 3 (2023), 524–538.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] https://arxiv.org/abs/1512.03385

[12] Yan He, Kebin Jia, and Zhihao Wei. 2023. Improvements in Forest Segmentation Accuracy Using a New Deep Learning Architecture and Data Augmentation Technique. *Remote Sensing* 15, 9 (2023). https://www.mdpi.com/2072-4292/15/9/2412

[13] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. arXiv:1905.02244 [cs.CV] https://arxiv.org/abs/1905.02244

[14] Nyo Me Htun, Toshiaki Owari, Satoshi Tsuyuki, and Takuya Hiroshima. 2024. Detecting Canopy Gaps in Uneven-Aged Mixed Forests through the Combined Use of Unmanned Aerial Vehicle Imagery and Deep Learning. *Drones* 8, 9 (2024), 484.

[15] Sambhav R. Jain, Albert Gural, Michael Wu, and Chris H. Dick. 2020. Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks. arXiv:1903.08066 [cs.CV] https://arxiv.org/abs/1903.08066

[16] Alexander Kmoch, Oleksandr Matsibora, Ivan Vasilyev, and Evelyn Uuemaa. 2022. Applied open-source discrete global grid systems. *AGILE: GIScience Series* 3 (2022), 41.

[17] Alexandre Lopes, Fernando Pereira dos Santos, Diulhio de Oliveira, Mauricio Schiezaro, and Helio Pedrini. 2024. Computer Vision Model Compression Techniques for Embedded Systems:A Survey. *Computers Graphics* 123 (2024), 104015. https://doi.org/10.1016/j.cag.2024.104015

[18] Sidi Ahmed Mahmoudi, Maxime Gloesener, Mohamed Benkedadra, and Jean-Sébastien Lerat. 2025. Edge AI System for Real-Time and Explainable Forest Fire Detection Using Compressed Deep Learning Models. *Proceedings Copyright* 847 (2025), 854.

[19] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. 2018. Attention U-Net: Learning Where to Look for the Pancreas. arXiv:1804.03999 [cs.CV] https://arxiv.org/abs/1804.03999

[20] Fernando Reboredo. 2013. Socio-economic, environmental, and governance impacts of illegal logging. *Environment Systems and Decisions* 33, 295–304.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597 [cs.CV] https://arxiv.org/abs/1505.04597

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (Eds.), Vol. 9351. Springer International Publishing, Cham, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

[23] SpatiaLite Project. 2025. Tessellations 4.0 – libspatialite. https://www.gaia-gis.it/fossil/libspatialite/wiki?name=tesselations-4.0. Accessed: 2025-06-28.

[24] Şule Nur Topgül, Elif Sertel, Samet Aksoy, Cem Ünsalan, and Johan E. S. Fransson. 2025. VHRTrees: a new benchmark dataset for tree detection in satellite imagery and performance evaluation with YOLO-based models. *Frontiers in Forests and Global Change* Volume 7 - 2024 (2025). https://doi.org/10.3389/ffgc.2024.1495544

[25] Sik-Ho Tsang. 2018. *Review: DeepLabv3+—Atrous Separable Convolution (Semantic Segmentation)*. https://sh-tsang.medium.com/review-deeplabv3-atrous-separable-convolution-semantic-segmentation-a625f6e83b90 Medium blog; accessed 2025-06-29.

[26] Josh Veitch-Michaelis, Andrew Cottam, Daniella Schweizer, Eben N. Broadbent, David Dao, Ce Zhang, Angelica Almeyda Zambrano, and Simeon Max. 2024. OAM-TCD: A globally diverse dataset of high-resolution tree cover maps. arXiv:2407.11743 [cs.CV] https://arxiv.org/abs/2407.11743

[27] Jisheng Xia, Yutong Wang, Pinliang Dong, Shijun He, Fei Zhao, and Guize Luan. 2022. Object-Oriented Canopy Gap Extraction from UAV Images Based on Edge Enhancement. *Remote Sensing* 14, 19 (2022). https://doi.org/10.3390/rs14194762

[28] Jiahong Xiang, Zhuo Zang, Xian Tang, Meng Zhang, Panlin Cao, Shu Tang, and Xu Wang. 2024. Rapid Forest Change Detection Using Unmanned Aerial Vehicles and Artificial Intelligence. *Forests* 15, 9 (2024). https://doi.org/10.3390/f15091676

[29] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. 2018. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. arXiv:1807.10165 [cs.CV] https://arxiv.org/abs/1807.10165

A EVALUATION TABLES

A.1 Evaluation Tables - Performance Measuring with Dynamic Thresholding

Table 7. Tiny DeepLabV3+ performance for different thresholds

Thr	Loss	Acc	F1 Score	IoU	Recall	Precision
0.400	0.2299	0.9059	0.8834	0.8091	0.9161	0.8676
0.425	0.2299	0.9068	0.8835	0.8094	0.9109	0.8725
0.450	0.2299	0.9076	0.8833	0.8093	0.9056	0.8773
0.475	0.2299	0.9081	0.8829	0.8089	0.9002	0.8819
0.500	0.2299	0.9084	0.8826	0.8085	0.8946	0.8863
0.525	0.2299	0.9085	0.8817	0.8074	0.8888	0.8907
0.550	0.2299	0.9084	0.8805	0.8059	0.8828	0.8949
0.575	0.2299	0.9081	0.8789	0.8040	0.8765	0.8989
0.600	0.2299	0.9076	0.8770	0.8016	0.8698	0.9028
0.625	0.2299	0.9069	0.8752	0.7992	0.8628	0.9068
0.650	0.2299	0.9059	0.8726	0.7959	0.8555	0.9105
0.675	0.2299	0.9047	0.8696	0.7920	0.8475	0.9143
0.700	0.2299	0.9032	0.8661	0.7875	0.8391	0.9182

Table 8. Tiny Unet performance for different thresholds

Thr	Loss	Acc	F1 Score	IoU	Recall	Precision
0.400	0.2977	0.8528	0.8392	0.7480	0.9457	0.7767
0.425	0.2977	0.8585	0.8426	0.7533	0.9395	0.7863
0.450	0.2977	0.8645	0.8462	0.7586	0.9315	0.7975
0.475	0.2977	0.8702	0.8491	0.7631	0.9219	0.8096
0.500	0.2977	0.8750	0.8510	0.7663	0.9111	0.8217
0.525	0.2977	0.8787	0.8517	0.7680	0.8994	0.8332
0.550	0.2977	0.8814	0.8511	0.7680	0.8867	0.8438
0.575	0.2977	0.8829	0.8491	0.7661	0.8730	0.8538
0.600	0.2977	0.8835	0.8456	0.7624	0.8581	0.8625
0.625	0.2977	0.8831	0.8409	0.7572	0.8422	0.8716
0.650	0.2977	0.8816	0.8348	0.7501	0.8249	0.8797
0.675	0.2977	0.8791	0.8271	0.7410	0.8063	0.8885
0.700	0.2977	0.8753	0.8177	0.7299	0.7863	0.8970

Table 9. Tiny SegNetV2 performance for different thresholds

Thr	Loss	Acc	F1 Score	IoU	Recall	Precision
0.400	0.3694	0.8537	0.8064	0.7098	0.8127	0.8399
0.425	0.3694	0.8530	0.8005	0.7038	0.7967	0.8484
0.450	0.3694	0.8516	0.7937	0.6967	0.7806	0.8553
0.475	0.3694	0.8496	0.7860	0.6888	0.7634	0.8602
0.500	0.3694	0.8465	0.7760	0.6783	0.7441	0.8670
0.525	0.3694	0.8418	0.7630	0.6645	0.7214	0.8736
0.550	0.3694	0.8363	0.7489	0.6495	0.6986	0.8766
0.575	0.3694	0.8305	0.7347	0.6346	0.6773	0.8776
0.600	0.3694	0.8245	0.7204	0.6196	0.6568	0.8765
0.625	0.3694	0.8177	0.7053	0.6037	0.6362	0.8763
0.650	0.3694	0.8101	0.6888	0.5864	0.6146	0.8758
0.675	0.3694	0.8017	0.6708	0.5678	0.5920	0.8719
0.700	0.3694	0.7922	0.6511	0.5476	0.5681	0.8689

A.2 Evaluation Tables - HyperParameter Grid Search

Table 10. Validation Accuracy for different combinations of Batch Size and Learning Rates for the Tiny DeepLabV3+ Model using a Grid Search approach

Learning Rate	Batch Size	Validation Accuracy
0.001	2	0.7952
0.001	4	0.8539
0.001	8	0.8175
0.001	16	0.8504
0.001	32	0.8318
0.0001	2	0.8114
0.0001	4	0.8240
0.0001	8	0.8100
0.0001	16	0.8275
0.0001	32	0.8104
1e-05	2	0.7639
1e-05	4	0.7873
1e-05	8	0.7960
1e-05	16	0.7915
1e-05	32	0.7886

Table 11. Validation Accuracy for different combinations of Batch Size and Learning Rates for the Tiny SegNetV2 Model using a Grid Search approach

Learning Rate	Batch Size	Validation Accuracy
0.001	2	0.7753
0.001	4	0.6862
0.001	8	0.7648
0.001	16	0.7574
0.001	32	0.7833
0.0001	2	0.7538
0.0001	4	0.7428
0.0001	8	0.7099
0.0001	16	0.6500
0.0001	32	0.6051
1e-05	2	0.6813
1e-05	4	0.6397
1e-05	8	0.6210
1e-05	16	0.5587
1e-05	32	0.4539

Table 12. Validation Accuracy for different combinations of Batch Size and Learning Rates for the Tiny SegNetV2 Model using a Grid Search approach

Learning Rate	Batch Size	Validation Accuracy
0.001	2	0.7990
0.001	4	0.8371
0.001	8	0.7780
0.001	16	0.8444
0.001	32	0.8523
0.0001	2	0.8060
0.0001	4	0.7976
0.0001	8	0.7923
0.0001	16	0.8200
0.0001	32	0.7919
1e-05	2	0.5808
1e-05	4	0.6803
1e-05	8	0.7163
1e-05	16	0.5489
1e-05	32	0.5268

B MODEL PREDICTION VISUALIZATIONS - INPUT IMAGE, GROUND TRUTH, GREYSCALE PREDICTION, BINARIZED PREDICTION

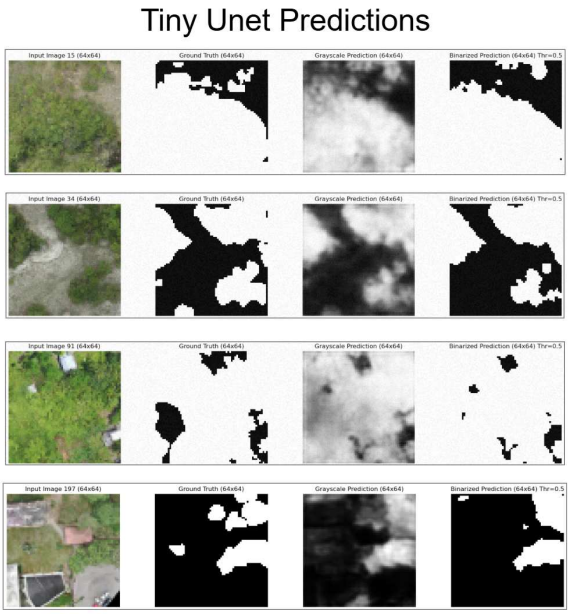

Fig. B.2. Examples of predictions done using Tiny Unet on 4 images. Each row contains the input image, the ground truth, the greyscale prediction and the binarized prediction using the optimal threshold (0.5)

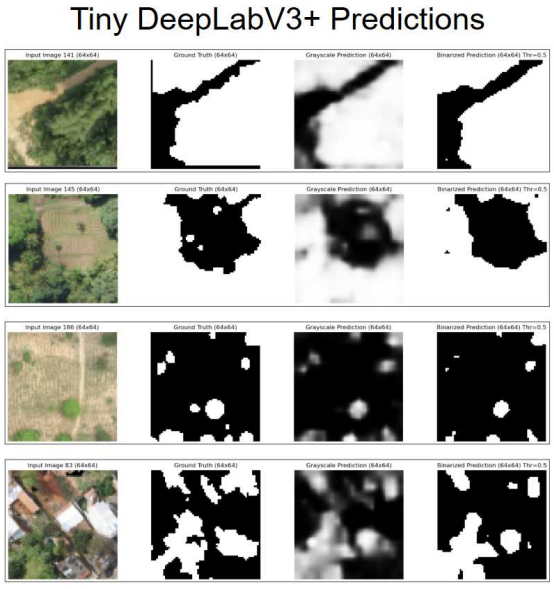

Fig. B.1. Examples of predictions done using Tiny DeepLabV3+ on 4 images. Each row contains the input image, the ground truth, the greyscale prediction and the binarized prediction using the optimal threshold (0.5)

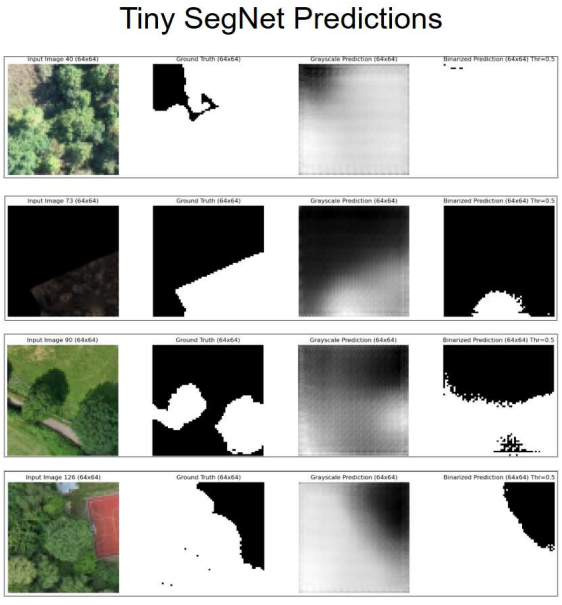

Fig. B.3. Examples of predictions done using SegNetV2 on 4 images. Each row contains the input image, the ground truth, the greyscale prediction and the binarized prediction using the optimal threshold (0.5)

C LEARNING CURVES - TRAINING VS VALIDATION ACCURACY METRICS

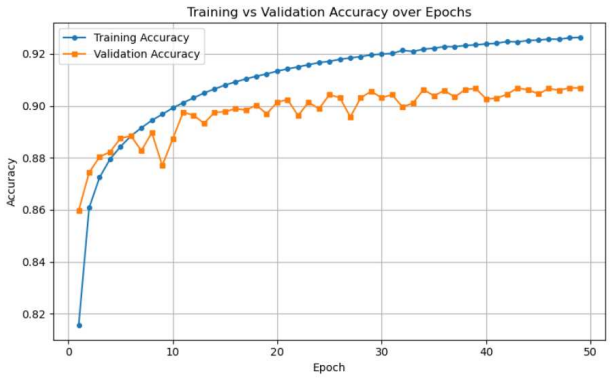

Fig. C.1. Training vs Validation Accuracy during training for Tiny DeepLabV3+

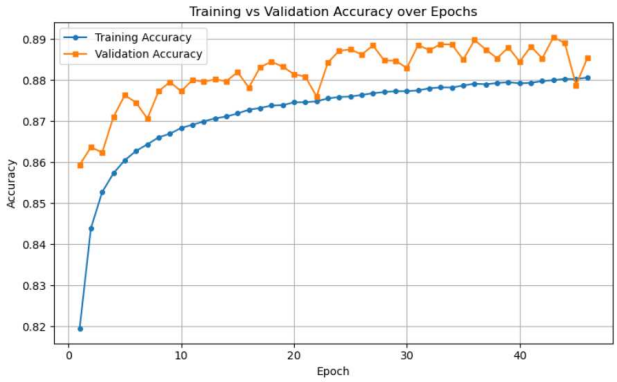

Fig. C.2. Training vs Validation Accuracy during training for Tiny Unet

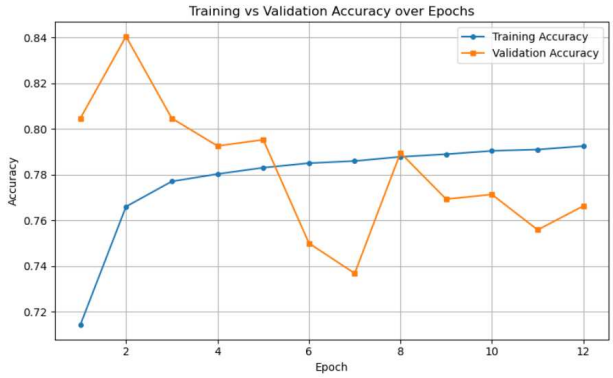

Fig. C.3. Training vs Validation Accuracy during training for Tiny SegNetV2