Detection of Buildings

Greta Virpšaitė
Software Engineering, 3rd year
Faculty of Mathematics and Informatics
Vilnius, Lithuania
greta.virpsaite@mif.stud.vu.lt

Andrius Šukys
Software Engineering, 3rd year
Faculty of Mathematics and Informatics
Vilnius, Lithuania
andrius.sukys@mif.stud.vu.lt

Benas Skripkiūnas
Software Engineering, 3rd year
Faculty of Mathematics and Informatics
Vilnius, Lithuania
benas.skripkiunas@mif.stud.vu.lt

***Abstract*—In this study, the challenge of accurate building recognition from satellite and aerial imagery is addressed. *Open Buildings* dataset from a region in South Sudan and *Google Maps* satellite view focusing on the United States and Europe are used. Methodology includes analyzing detection thresholds using U-Net architecture with both common *ImageNet* and manually set (the model is trained from scratch) weights. Performance of the models is evaluated using Dice coefficient and Intersection over Union (IoU) metrics. The results demonstrate that this approach effectively enhances segmentation accuracy, offering a solution for building detection in complex environments and points out major inconsistencies that could be analyzed in upcoming studies.**

***Index Terms*—Convolution, Convolutional Neural Networks (CNN), Semantic Segmentation, Mask, U-Net, Detection, Threshold, Fine-Tuning.**

I. INTRODUCTION

Correct recognition of buildings from satellite and aerial photography is an important task in urban planning and geographical information systems. The diversity in building architecture and the complexity of urban environments frequently provide challenges for traditional approaches. This study contributes by suggesting a technique that makes use of the *Open Buildings* dataset in addition to self-collected dataset of *Google Maps*, with an emphasis on places in the United States and Europe. The technique described in this study focuses on finding optimal thresholds when training a model with *ImageNet* and manually set weights to find the best outcome for both models and compare them.

II. METHODS

1) Tools:
   QGIS – an open-source Geographic Information System (GIS) software that enables users to analyze and edit spatial information on various maps and databases.
   Google Colaboratory (Pro) – a cloud-based *Python* programming environment that integrates *Google Drive* and supports collaborative data science tasks, facilitating easy access to computing resources. For this particular task, *T4 GPU* was used.
   PyTorch – an open-source machine learning library for *Python*.
   REST API – created using *Python*, the *API* interface is created to accept an image and return its predicted mask to evaluate the accuracy of different models.
2) Evaluation Metrics:
   To evaluate the segmentation model, *Dice* coefficient and *IoU* metrics were used. These metrics are often used together in applications that need precise segmentation, for example, in medical imaging, due to their usefulness in measuring overlap and ensuring accurate boundaries of needed figures. [3]
   Dice – Dice-Sørensen coefficient:

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{1}$$

   - Measures the overlap between two sets.
   - Ranges from 0 to 1, where 0 indicates no overlap and 1 indicates perfect overlap.
   - *Dice* emphasizes the overlap between two sets more than *IoU* because there is a *2* in the numerator, making it more sensitive to overlaps.
   - Practically the same as *F1* score in the context of binary classification.

   IoU – Intersection over Union:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

   - Measures the overlap between two sets divided by their union.
   - Ranges from 0 to 1, where 0 indicates no overlap and 1 indicates perfect overlap.
   - Direct measure of overlapping.
   - Also known as *Jaccard Index*.
3) Activation Function:

   Sigmoid function – used in neural networks for binary classification tasks.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

4) Loss Function:

   nn.BCEWithLogitsLoss – [1] This loss function combines a *Sigmoid* layer and the *BCELoss* in one single class. This version is more numerically stable than using a plain *Sigmoid* followed by a *BCELoss* as, by combining the operations into one layer, an advantage of the log-sum-exp trick for numerical stability is taken.

The unreduced (i.e., with `reduction` set to `none`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \tag{4}$$

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right], \tag{5}$$

where N is the batch size. If `reduction` is not `none` (default `mean`), then

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{mean;} \\ \operatorname{sum}(L), & \text{if reduction} = \text{sum.} \end{cases} \tag{6}$$

5) Weights:
Model 1 uses common ***ImageNet*** weights
- mean = [0.485, 0.456, 0.406],
- std = [0.229, 0.224, 0.225].

Model 2 uses **manually set** weights of
- mean = [0, 0, 0],
- std = [1, 1, 1].

6) Optimizer:
Adam – an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. The decision to use this optimizer in the study was taken because [2] empirical results demonstrate that *Adam* works well in practice and compares favorably to other stochastic optimization methods.

7) Architecture:
U-Net [5] – In the mentioned paper as seen in Figure 1, the *U-Net* architecture consists of two paths – contracting and expanding. The contracting path consists of two 3x3 convolutions followed by *ReLU* activation and a 2x2 *max pooling* for downsampling, with each step doubling the number of feature channels. The expansive approach involves upsampling the feature map, a 2x2 convolution that reduces the number of feature channels, concatenation with cropped feature maps from the contracting path, and two 3x3 convolutions followed by *ReLU* activation. The final layer applies a 1x1 convolution to convert the feature vector to the required number of channels.
The *U-Net* architecture that this study uses is similar to the one mentioned above. It includes a series of convolutional blocks for downsampling and upsampling paths. The key differences are in the use of batch normalization in implementation, the explicit definition of upsampling and downsampling blocks, and the use of center cropping to handle dimensional mismatches during concatenation. The architecture was changed in order to improve training stability and segmentation accuracy. The implementation features:
- *DoubleConvolution* block: Applies two 3x3 convolutions with padding, batch normalization, and *ReLU* activation.
- *DownSample* block: Uses *max pooling* followed by the *DoubleConvolution* block.
- *UpSample* block: Uses transposed convolutions for upsampling and concatenates the upsampled feature maps with the corresponding feature maps from the contracting path.
- *Final layer*: Uses a 1x1 convolution to produce the final output.

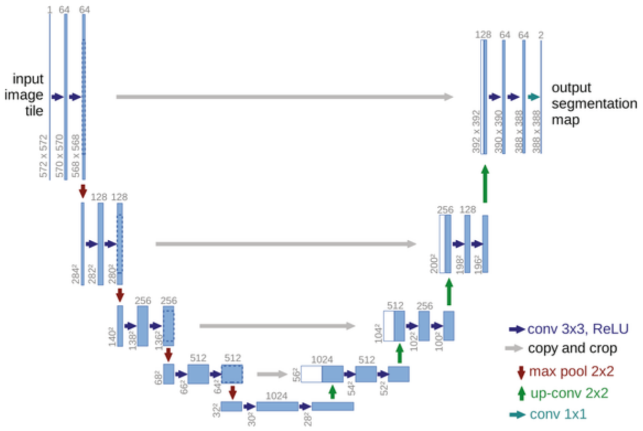


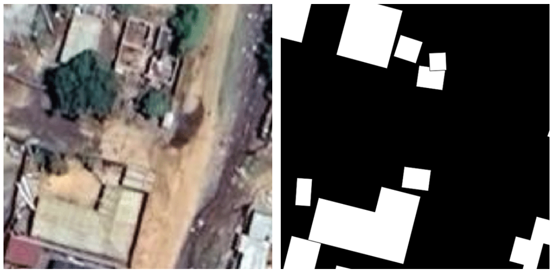
Figure 1 – U-Net Architecture

In Figure 1, each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The *x-y* size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [5]

III. DATASET

The dataset that was utilized in training was a fraction of the *Open Buildings* dataset (*171_buildings.csv.gz*), which includes satellite imagery from South Sudan. To optimize the machine learning model, images were separated into different sets, assigning 1,200 (~87%) for training purposes, and 177 (~13%) images for validation to scale the model's accuracy and generalization capabilities over data that has not yet been met. Additionally, the study uses custom dataset with 50 handpicked images from *Google Maps*, containing locations in the US and Europe, to serve as a separate test set.

In Figure 2, an image and its corresponding mask from training dataset is shown as an example.



Image

Mask

Figure 2 – An example image and its mask

To further enhance model performance and ensure it generalizes well across various conditions, several image augmentation techniques [4] were implemented during the training phase. Using the *Albumentations* library, training data went through various transformations: resizing to 224x224 pixels to maintain consistency in input size, random rotations with a limit of ±10 degrees to simulate different viewing angles, and horizontal flips with a 50% probability to increase the diversity of training perspectives. These transformations help mimic real-world variations and improve the model in a way where it is better protected against overfitting. Additionally, normalization was applied to standardize the pixel values across all images. For **Model 1**, mean of [0.485, 0.456, 0.406] and standard deviation of [0.229, 0.224, 0.225] was used. For **Model 2**, mean of [0, 0, 0] and standard deviation of [1, 1, 1] was used. The validation set received similar preprocessing steps, except for the random transformations, to preserve the original integrity of the data for accurate performance evaluation.

IV. RESULTS

The outcomes analyzed in this section are available on GitHub (includes source codes, test environment and this report) and Google Drive (includes used datasets, trained models and predictions).

Training Process

Before analyzing the actual performance of the aforementioned models (**Model 1** and **Model 2**), training process must be considered. As **Model 1** had *ImageNet* weights assigned and **Model 2** was trained from scratch, their training process slightly differed. It can be observed in *Training and Validation losses* graphs shown in Figure 3 (for **Model 1**) and Figure 4 (for **Model 2**).

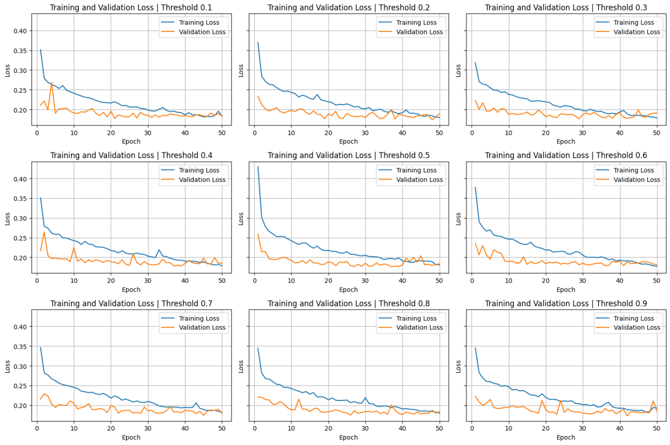


Figure 3 – Training and Validation losses of Model 1

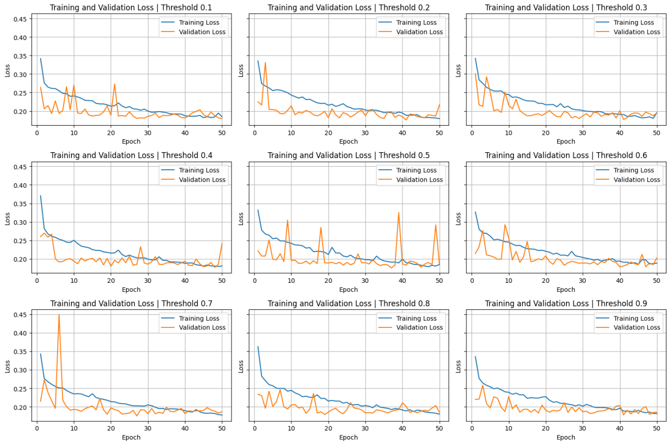


Figure 4 – Training and Validation losses of Model 2

As seen in Figure 3, the model with assigned *ImageNet* weights performed in a relatively stable fashion when looking at validation loss. There are minor spikes in the mentioned graph but none of them should be considered major compared to training process of **Model 2**. Validation loss of **Model 2** (which was trained from scratch) pictured in Figure 4 spiked quite often, namely at thresholds from 0.4 to 0.7. There are some possible causes explaining this phenomenon, such as instability of training process because of used weights of mean = [0, 0, 0] and std = [1, 1, 1], or a learning rate that was inappropriate for the model at that stage.

However, both models have shown gradually decreasing training loss throughout the process, except for the initial epochs where the losses dropped significantly (which indicates that the models were learning). The gaps between training and validation losses are for the most part small and stable. Looking at the final epochs, losses are close to plateauing, so a conclusion was made that the models were trained for an appropriate amount of time but could have improved their performance if the training period went on for slightly longer than 50 epochs.

Threshold Analysis

One of the main aims of the study was to perform threshold analysis on both differently trained models. Figures 5-13 display the actual image, as well as the differences between true masks and predictions of **Model 1** and **Model 2** for each threshold going from 0.1 to 0.9 with a step size of 0.1.

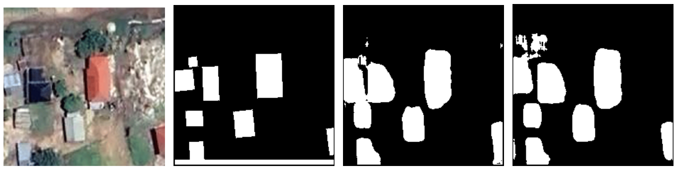


Image True Mask Model 1 Model 2

Figure 5 – Threshold of 0.1

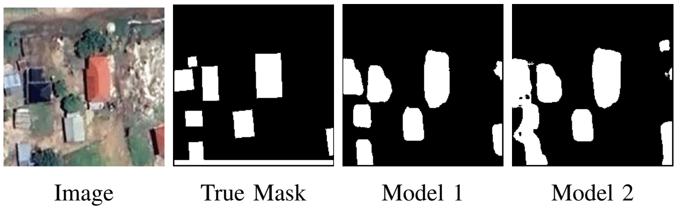


Figure 6 – Threshold of 0.2

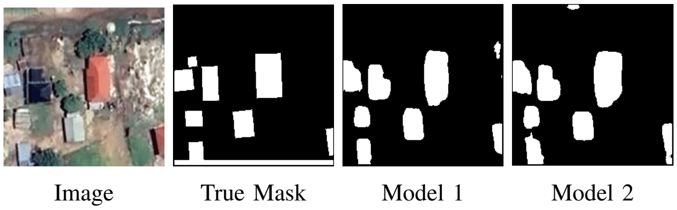


Figure 7 – Threshold of 0.3

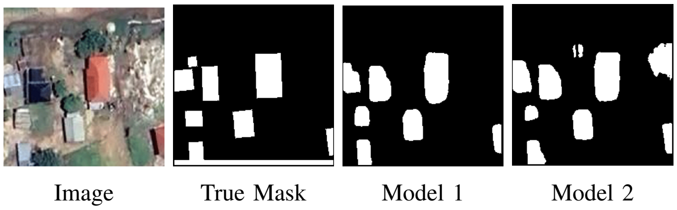


Figure 8 – Threshold of 0.4

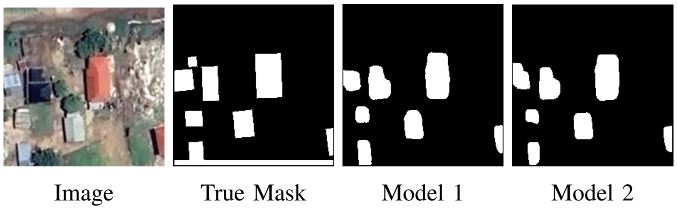


Figure 9 – Threshold of 0.5

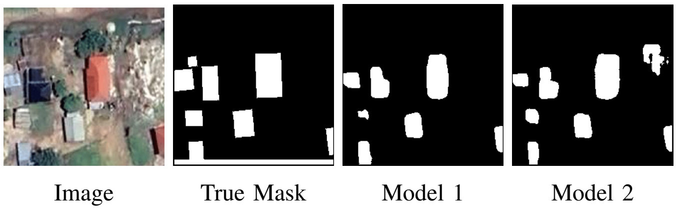


Figure 10 – Threshold of 0.6

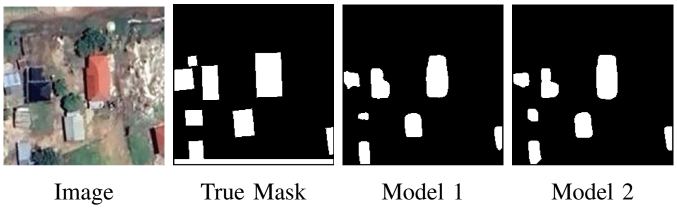


Figure 11 – Threshold of 0.7

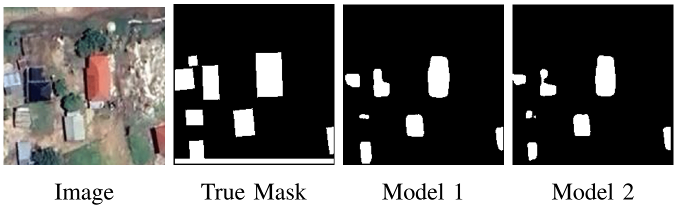


Figure 12 – Threshold of 0.8

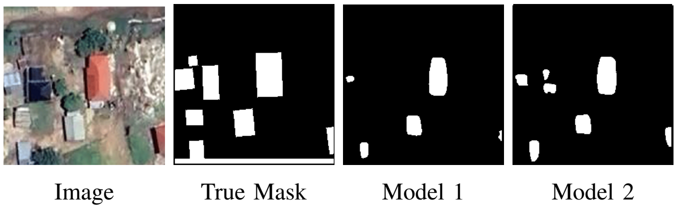


Figure 13 – Threshold of 0.9

As expected, thresholds nearing the lower end (getting close to 0.1) of the spectrum recognize buildings where they are not present, likewise, thresholds nearing the higher end (getting close to 0.9) omit some of the buildings entirely. Taking the nature of thresholds into consideration, this result was expected.

However, this section only provides visual representation of how both models perform at different thresholds. Such a perspective suggests a quick and subjective overview rather than an accurate judgement. To objectively evaluate both models at all thresholds, *Dice* and *IoU* metrics were used.

Determination of Best Thresholds for Models

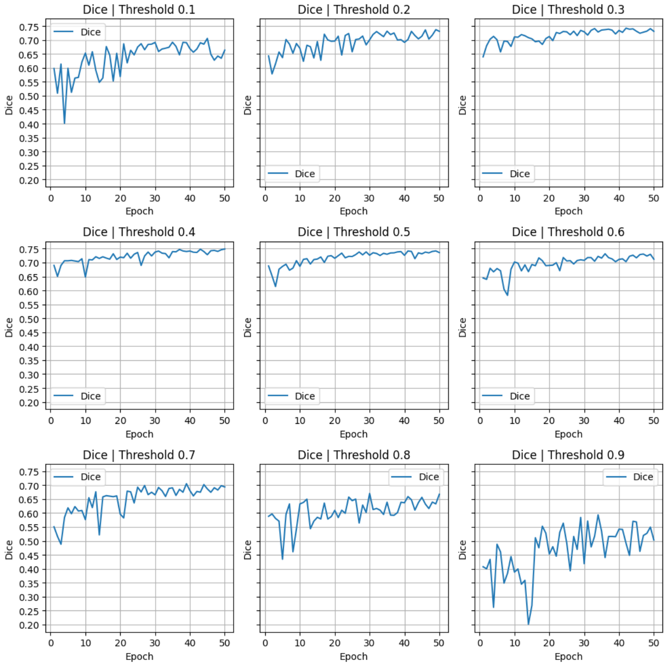


Figure 14 – Dice of Model 1

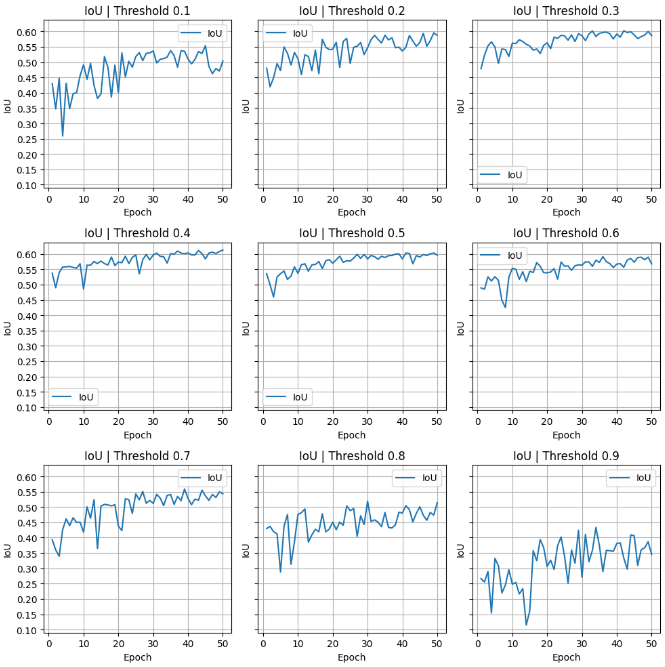


Figure 15 – IoU of Model 1

According to *Dice* and *IoU* values calculated for each threshold, **Model 1** peaks at **Dice of 0.7486** (**IoU = 0.6128**) at **threshold = 0.4**, as shown in Figure 14 and Figure 15.

Both *Dice* and *IoU* metrics evaluate overlap of predicted and actual masks so they are tightly correlated. Thus, the best threshold could have been determined by using only one of them interchangeably. In this study, *Dice* was used to determine the best threshold for a model.

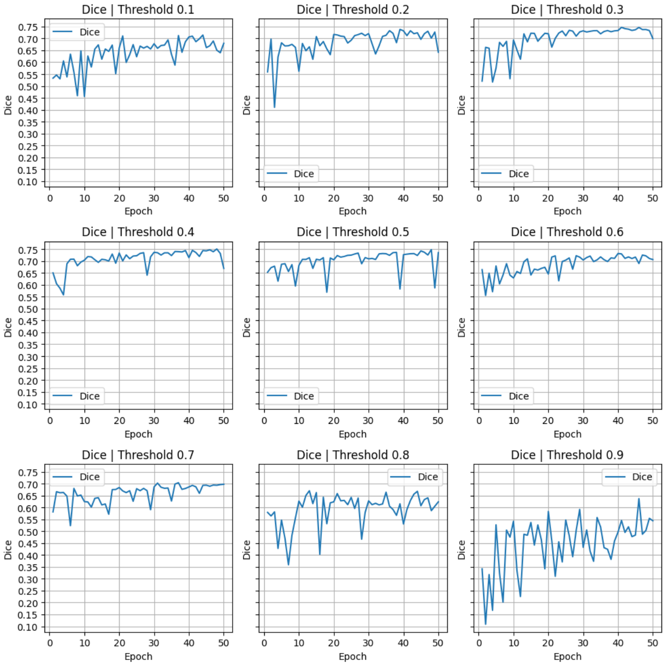


Figure 16 – IoU of Model 2

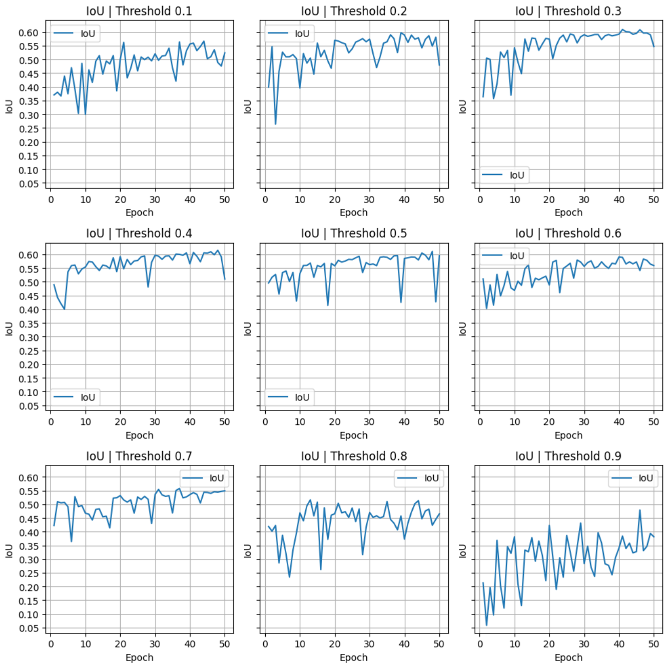


Figure 17 – IoU of Model 2

Model 2 peaks at **Dice of 0.7062** (**IoU = 0.5592**) at **threshold = 0.6**), as shown in Figure 16 and Figure 17.

As the best-performing thresholds for both models were determined, they could be used with their respective models to make a comparison whether using *ImageNet* weights and fine-tuning the model or training the model from scratch is better in practice.

Testing of Models with self-collected Google Maps Dataset

Using a self-created environment that allows to test models by using a *REST API* interface, both models (trained on their respective best thresholds) were uploaded and tested with a custom dataset that was not used while training. This chapter is an overview of testing results, shown in Figures 18-23.

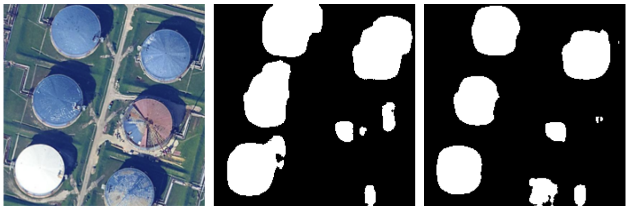


Image Model 1 Model 2

Figure 18 – *Google Maps* Dataset Prediction No. 1

Model 1 which had better *Dice* values (at **0.7486**, compared to **0.7062** of **Model 2**) does not always output visibly better results, as shown in Figure 18. Seemingly, it is less receptive to shadows and recognizes round shapes better. The same tendency can be seen in Figure 18 where **Model 2** is more capable of identifying elongated shapes successfully than **Model 1**.

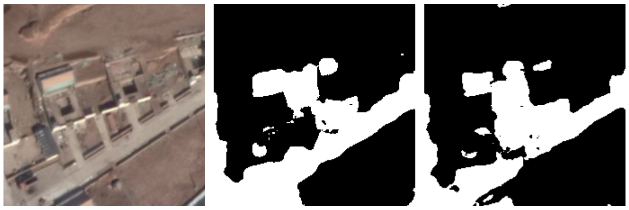


Image Model 1 Model 2

Figure 19 – *Google Maps* Dataset Prediction No. 2

Looking at Figure 19, it is evident that the models struggle to identify buildings in images with shadows and also face challenges in accurately detecting buildings in images that include roads too. The main proposed reason for these inconsistencies is that the colors of roads and shadows in the photographs are often similar to those of roofs. To combat these faulty detections, an additional study would be required and should remain as a proposed continuation of this document.

However, often the models accurately detected buildings, indicating that these occasional faulty detections should not overshadow their overall performance. Some of these successful detections are shown in Figures 20-23.

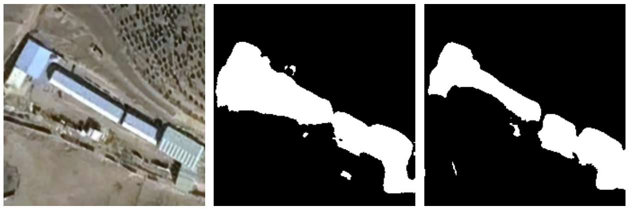


Image Model 1 Model 2

Figure 20 – *Google Maps* Dataset Prediction No. 3

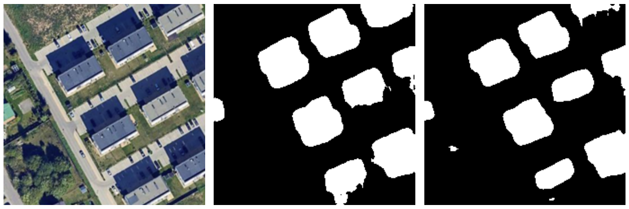


Image Model 1 Model 2

Figure 21 – *Google Maps* Dataset Prediction No. 4

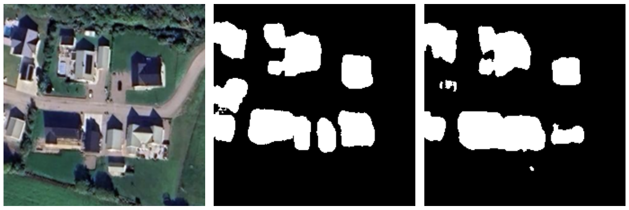


Image Model 1 Model 2

Figure 22 – *Google Maps* Dataset Prediction No. 5

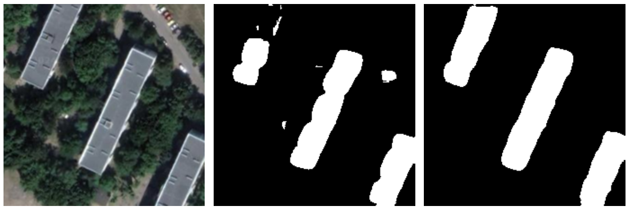


Image Model 1 Model 2

Figure 23 – *Google Maps* Dataset Prediction No. 6

V. Takeaways

- **Potential Improvements.** Small dataset size is a possible reason for slight underfitting. Increasing the size of data or training the model for more epochs could improve the training process for more successful results. Additionally, more extensive data augmentations could enhance the model's accuracy.
- **Performance.** The results show that while both weight sets behave differently, the model using *ImageNet* weights trains more smoothly than the one that was trained from scratch.
- **Theory versus Practice.** *Dice* value stating that **Model 1** should have been objectively better than **Model 2**, appeared to be misleading in some test cases and did not reflect actual performance.
- **Challenges.** Shadows and roads were sometimes misclassified as buildings. It could be a subject for upcoming studies in this field.

References

[1] PyTorch Documentation. *torch.nn.BCEWithLogitsLoss*, 2023.

[2] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[3] Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. Towards a guideline for evaluation metrics in medical image segmentation. *BMC Research Notes*, 15(1):96, 2022.

[4] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. 12 2017.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany*, 2015.