CIL Project Report – Road Segmentation

Alexey Gavryushin, Mateusz Nowak, Anne Marx, Noureddine Gueddach
Group: StackMoreLayers
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Roads are at the center of human mobility, communication, and civilization, with an enormous number of kilometers built around the globe. Manually segmenting them, e.g. to create digital maps, has become a tremendous and expensive task. Yet, with the increasing availability of satellite images, it is now possible to ease this task using Machine Learning tools. In this paper, we propose a new densely connected architecture which we call *U-Net Exp*. We compare our model to several baselines on a benchmark dataset to further showcase the effect of techniques such as ensembling and transfer learning, which enormously improved the performance of our models. Finally, we discuss a novel segmentation approach with road connectivity prior, based on Reinforcement Learning.**

I. Introduction

Convolutional neural networks (CNNs) are one of the most well-known and successful types of model architecture in image segmentation. One of the first architectures specifically tailored toward this task was the pyramidically-structured U-Net [1], which achieved state-of-the-art medical image segmentation results in 2016. Since then, a plethora of derivative models have been presented, including UNet++ [2], Attention U-Net [3], U^2-Net [4], and ResUNet [5]. In this work, we propose a novel, more densely connected architecture - *U-Net Exp*.

Whereas these models are trained under supervision, to our knowledge - there is no research in the field of reinforcement learning (RL) for road segmentation. Similar disciplines such as medical image segmentation have priorly used RL to mark regions of interest and refine the segmentation ([6]) which achieved higher than state-of-the-art performance. We, therefore, propose a novel RL architecture with additional built-in segmentation connectivity to avoid disconnected segmentations which are unwanted artefacts of supervised deep neural networks (DNN).

The structure for the remainder of this report is as follows: In Section II, we describe how we extended the original data both by acquiring new images and augmenting existing ones. Section III outlines the baseline models used to compare our results against and shows our contributions before presenting the results of our experiments in Section IV. We discuss our findings in Section V and summarize the main ideas of this paper in Section VI. Further details are given in the Appendix.

II. Extending the dataset

Our benchmark dataset, provided as part of a Kaggle competition [7], contains 144 images, which is inadequate to train a robust model that can generalize well to unseen data. To solve this problem, we resort to two different approaches while leaving the images' resolution unchanged. Using these techniques, we expand the dataset to contain 11179 training images.

A. Augmentation of the old data

We augment each image by randomly applying geometric transformations such as horizontal and vertical flips, rotations, translations, and scaling, as well as lighting transformations - changing the contrast, luminosity, and saturation. These transformations allow us to produce new, plausible training images comparable to the existing samples that simulate different environmental conditions in the images. Two variants of this technique are tested, with varying results. The first creates new data through the mentioned augmentations once and saves them. After that, we use the newly generated and original images for training without further transformations. The second variant creates new data on-the-fly by loading an image and altering it anew in the previously described way before passing it to the model.

B. Obtaining new data

The second approach focuses on acquiring new training data. The biggest challenge is to obtain images comparable in distribution to the original dataset so as not to skew the total dataset's distribution in the direction of the newly gathered data, worsening the model's performance on the benchmark. We carefully investigate the original dataset and conclude that it consists of images likely retrieved from the two US cities of Boston, MA, and Los Angeles, CA. We then automatically scrape an additional 4335 images from these regions and hand-filter them to 1622 in order to keep those that we consider similar to the benchmark samples. Some of the extracted data are visualized in the APPENDIX.

III. MODELS AND METHODS

For this project, we build a cross-platform framework[1] capable of handling both PyTorch and TensorFlow models, with an MLflow[8] backend to store, share and monitor our experiments simultaneously. The framework helps us get better insights into our models by logging necessary details such as codebase snapshots, command-line arguments, hyper-parameters used for different training runs, checkpoints, and model performances.

A. Baselines

To evaluate our models, we compare them to four baselines, namely:

1) **U-Net3+ [9]:** The U-Net variant on top of which our *U-Net Exp* is built
2) **DeepLabV3 [10]:** A commonly used segmentation baseline employing atrous convolution[11]
3) **SegFormer [12]:** a Transformer [13]-based segmentation network
4) **Lawin [14]:** An extension to SegFormer leveraging a *large window attention module* inspired by atrous convolutions in a newly designed *large window head*

[1]https://github.com/Futuramistic/CIL2022

More details about the baseline network architectures can be found in the APPENDIX.

B. U-Net Exp

U-Net Exp is created based on the architecture of UNet3+ [9]. We propose new inter-skip connections that propagate the activation signal from one part of the network to the other. Additionally, UNet3+ adds inter-skip connections to its decoder part. We found in our experiments that similar links within the encoder structure are beneficial. We believe we can make more accurate predictions without further extending the decoder of UNet3+ by creating a richer representation within its encoder.

The new inter-skip connection gathers information from past layers and convolves each layer's result to assemble it into a new input to the existing UNet decoder nodes - X^i_{De}. On the encoder side, we have used the proposed UNet3+ architecture, which collects information from previous decoder layers and the outputs from the higher-dimension representation from the encoder layer. The only irregularity occurs within the first encoder layer - the first encoder node's result equals the last decoder node's. An inter-skip connection can be represented by:

$$X^i_{De} = \begin{cases} X^i_{En} & \text{, if } i = N \\ \tilde{X}^i_{De} & \text{, otherwise} \end{cases}$$

$$\tilde{X}^i_{De} = C\{C(D(X^k_{En}))^{i-1}_{k=1}, C(X^i_{De}, C(U(X^k_{De}))^N_{k=i+1}\},$$

Equation 1: Decoder inter-skip connection

where $C(\cdot)$ denotes a convolution block, $D(\cdot)$ and $U(\cdot)$ - downsampling and upsampling respectively.

Each inter-skip connection convolves each layer's input with a convolution that uses 64 filters. Then, all inputs are concatenated and convolved yet again, but this time - using 320 filters, considering all the given data. In UNet3+ architecture, this convolution replaces the original UNet decoder node (see Figure 6 in the APPENDIX).

We perform similar steps on the encoder side of the network (see Equation **??**). For each encoder node X^i_{En}, we convolve all the previous encoder nodes' outputs, except the most recent one, with 64 filters each. Then, the network convolves the information from the most recent node with the number of filters equal to its channels. After that, we concatenate this information and feed it into the standard UNet encoder node X^i_{En}, with the number of filters adjusted by the previous block's size, allowing all the relevant information to be accounted for by the node. Additionally, using these types of connections, we can seamlessly combine information from other pre-trained networks by adding respective layers to the encoder inputs with the new architecture. The only exception is present within the first decoder node - it works as a standard UNet decoder node, convolving the input.

$$X^i_{Ee} = \begin{cases} C(In) & \text{, if } i = 1 \\ C\{C(D(X^k_{En}))^{i-1}_{k=1}\} & \text{, otherwise,} \end{cases} \tag{1}$$

Equation 2: Encoder inter-skip connection

Moreover, we change the ReLU activation function to the SiLU [15] activation function (SiLU$(x) = x\ \sigma(x)$) within our architecture.

SiLU alleviates the vanishing gradient problem, which can occur given our deep architecture. Furthermore, distinguishing between negative values within our network might prove beneficial as the lack of ReLU's activation provides undesired information. The fact that there is no road should negatively affect other pixels within some proximity with an identifiable degree, which SiLU offers impeccably. However, ReLU outputs a zero signal, which can be associated with no contribution to the current prediction. ReLU's behavior, in this instance, can be deceiving to our network, as pixels within the middle of a field should carry different information between themselves compared to the pixels next to a road.

The network is also augmented and tested with different pre-trained models. The VGG [16] provides a better initial gradient and stabilizes the learning, allowing us to achieve satisfactory results quicker. However - it inhibits learning capabilities, limiting the network's results.

C. *RL Segmentation*

Segmentations predicted by supervised networks typically include artifacts, e.g. some parts of the segmentation end up disconnected from the others. Roads are usually straight, continuous and have the same width, even though they might be partially occluded by different objects, such as cars and vegetation. Based on this observation, we attempt a novel Reinforcement Learning architecture with built-in connectivity prior. Instead of creating a one-shot segmentation for the whole input image, we let an agent "walk" on the image. At each small step, it decides in which direction it should walk, whether to paint the pixels in the agent's current position, and if yes - the agent selects the appropriate "brush" size. We cast our objective as an RL optimization problem and solve it using policy gradients. We present more details (and issues) with this approach within the APPENDIX.

D. *Model Selection*

We perform hyper-parameter searches on our models using the HyperOpt [17] framework, which uses heuristics to optimize the search speed. Furthermore, we optimize our models' segmentation threshold both at train- and inference-time, finding the one maximizing the F1-score on our train set. Indeed, we have noticed that some models perform better with the prediction threshold slightly higher or lower than 0.5. Since the models we use contain parameters millions of parameters, they can easily overfit the limited data. To alleviate this problem, we perform 3-fold cross-validation during hyper-parameter tuning. After finishing the hyper-parameter selection, we perform a final search for the best models on three different splits of the whole training dataset. The best models of each split are then ensembled together.

E. *Ensembling*

To further improve our score, we propose two different ensemble approaches. Our first method consists of a simple ensembling of the rounded classifications of each network, using per-pixel majority voting. The second approach interprets the per-pixel output of a network as binary class probabilities. Averaging the predictions of all networks in the ensemble before rounding to either class allows us to account for the results of individual networks better and measure their confidence, leading to an even higher performance boost when compared to our initial approach.

F. Further optimizations

Further optimizations that empirically improve our results are transfer learning, dynamic sample weighting, inference-time data augmentation and using different loss functions. These methods are further described in the APPENDIX.

IV. RESULTS

We trained all baseline models on 3-fold splits, while doing a hyper-parameter search on learning rate and batch-size (where possible due to limited memory capacity). For each model, we select the checkpoint yielding the highest (weighted) F1-Score. The performance of the U-Net Exp network as compared to our baselines is shown in Table I.

	Metrics		
	Road F1	Macro F1	Weighted F1
U-Net3+	0.708	0.826	0.912
DeepLabV3	0.732	0.841	0.919
SegFormer	0.762	0.860	0.931
Lawin	**0.772**	**0.865**	**0.932**
U-Net Exp	0.727	0.838	0.920

Table I: Performance on the competition dataset.

We can observe from Table I that the Lawin network performed the best in comparison to other networks. Not only did it reach the highest F1 score, but also it surpassed all other metrics of various baselines. Although we have not exceeded the results of the heavier network like Lawin, our U-Net Exp **outperforms** its base model, the U-Net3+ and other similar ones, when they do not use any backbone.

Additional Table III within the Appendix depicts the impact of transfer learning on these models.

V. DISCUSSION

Even though our solution has not outperformed Lawin, it surpasses the other state-of-the-art models when these are not pre-trained. Given the minuscule number of parameters required to train (see Table II within the Appendix), our network seems to be a viable option that sacrifices a fraction of the performance for enormous gain within training space. We believe we opened many directions to explore within our results: Firstly, focusing more on the feature representation within the encoder can lead to further advances within the field. Many architectures overlook the encoder part. As we presented, the developments within feature extractions can lead to performance gains. Moreover, combining knowledge from different divisions of Machine Learning seems to hold potential and could expand the capabilities of neural networks.

VI. SUMMARY

In this project, we develop a novel ML framework and extend existing architectures by creating our U-Net version: the U-Net Exp. We dynamically vary segmentation thresholds, which can potentially boost the performance of other networks for other tasks. Moreover, we use techniques such as ensembling and transfer learning but also adapt them to the given problem. Finally, we proposed a novel RL-based architecture that presents some advantages over mainstream methods but also has some challenges and limitations in its current state.

Our model, the U-Net Exp, performs slightly worse when compared with Lawin. However, this performance gap pales in comparison when we account for models' sizes (see Tables II within the Appendix). We believe that, with a few tweaks, we can achieve exceptional results without drastically increasing the U-Net Exp's size. Furthermore, we believe the road segmentation domain can benefit from RL-based approaches, even without considerable success on our side. With suitable rewards, RL has a huge potential to easily enforce characteristics of the underlying data structure, such as consistent road width, continuity and straightness.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[2] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," 2018. [Online]. Available: https://arxiv.org/abs/1807.10165

[3] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert, "Attention u-net: Learning where to look for the pancreas," 2018. [Online]. Available: https://arxiv.org/abs/1804.03999

[4] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. Zaiane, and M. Jagersand, "U2-net: Going deeper with nested u-structure for salient object detection," vol. 106, 2020, p. 107404.

[5] F. I. Diakogiannis, F. Waldner, P. Caccetta, and C. Wu, "ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 162, pp. 94–114, apr 2020. [Online]. Available: https://doi.org/10.1016%2Fj.isprsjprs.2020.01.013

[6] Z. Tian, X. Si, Y. Zheng, Z. Chen, and X. Li, "Multi-step medical image segmentation based on reinforcement learning," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–12, 2020.

[7] (2022, July) Ethz cil road segmentation. [Online]. Available: https://www.kaggle.com/competitions/cil-road-segmentation-2022/data

[8] (2022, July) Mlflow: An open source platform for the machine learning lifecycle. [Online]. Available: https://mlflow.org/

[9] H. Huang, L. Lin, R. Tong, H. Hu, Q. Zhang, Y. Iwamoto, X. Han, Y.-W. Chen, and J. Wu, "Unet 3+: A full-scale connected unet for medical image segmentation," 2020. [Online]. Available: https://arxiv.org/abs/2004.08790

[10] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv:1706.05587*, 2017.

[11] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015. [Online]. Available: https://arxiv.org/abs/1511.07122

[12] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," 2021. [Online]. Available: https://arxiv.org/abs/2105.15203

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[14] H. Yan, C. Zhang, and M. Wu, "Lawin transformer: Improving semantic segmentation transformer with multi-scale representations via large window attention," 2022. [Online]. Available: https://arxiv.org/abs/2201.01615

[15] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2016. [Online]. Available: https://arxiv.org/abs/1606.08415

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[17] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[18] H. Schwenk and Y. Bengio, "Training methods for adaptive boosting of neural networks," in *NIPS*, 1997.

[19] N. Abraham and N. M. Khan, "A novel focal tversky loss function with improved attention u-net for lesion segmentation," 2018. [Online]. Available: https://arxiv.org/abs/1810.07842

Appendix

A. Baseline architectures

1) **U-Net3+:** This variation of U-Net uses additional skip connections going from each node of the encoder to every stage located below or at the same level in the decoder (See Figure 1). Additionally, in the decoder part, the features are not only fed to the directly succeeding decoding stage but also the subsequent ones. All this data is convolved and gathered as a final decoder node's input. Finally, the network uses deep supervision - the loss is on all intermediate stages of the decoder, treating each node's output as a separate network.
2) **DeepLabV3:** Proposed by Google in 2017 and successor of previous DeepLabV1 and DeepLabV2, DeepLabV3 uses a ResNet-101 as a feature extractor, extended with extra blocks using atrous convolutions (dilated convolutions, refer to Figure 2), helping broaden the field of view of the network, allowing to capture long-range context.
3) **SegFormer:** This new state-of-the-art architecture published in 2021, while still consisting of an encoder and a decoder, breaks away from previous approaches. The encoder uses a multi-resolution transformer architecture, while the encoder interestingly contains an MLP layer that fuses the extracted multi-scale features. Refer to Figure 3 for an overview of the architecture.
4) **Lawin:** This architecture shares some similarities with the SegFormer as it is also a Transformer and employs MLPs (See Figure 4). The Lawin network uses a Spatial Pyramid Pooling (SPP) scheme, similar to DeepLabV3, replacing the Atrous convolutions with so-called Large Window Attention, yielding the LawinSPP, which acts as the decoder part of the network.

B. Metrics

To report our results, we used the 'Road F1-Score', the 'Macro F1-Score', and the 'Weighted F1-Score'. Before describing each metric and explaining the differences, let's briefly review the F1-Score. Essentially, this score serves as a compromise between the precision ($P = \frac{TP}{TP+FP}$) and the recall ($R = \frac{TP}{TP+FN}$). Indeed, the precision can be maximal (=1) even for a model that never predicts positives. Similarly, we can achieve a recall of 1 can by having the model systematically predict positives. But to achieve a high F1-Score, both precision and recall should be high since its computation is the harmonic mean of the two:

$$\mathrm{F1} = 2 \times \frac{P * R}{P + R}$$

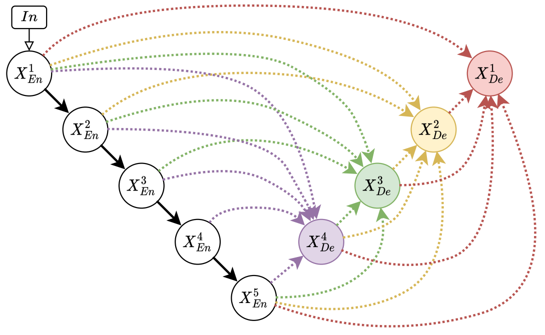


Figure 1: The U-Net3+ architecture

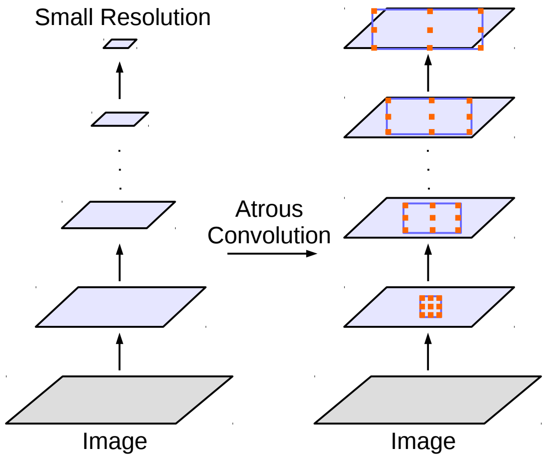


Figure 2: Illustration of the DeepLabV3's atrous convolution, allowing to preserve larger resolution

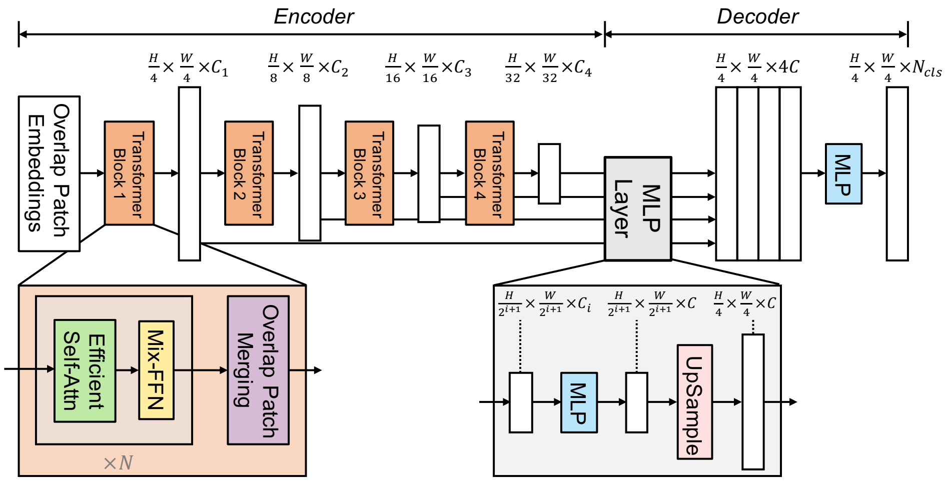


Figure 3: The SegFormer architecture

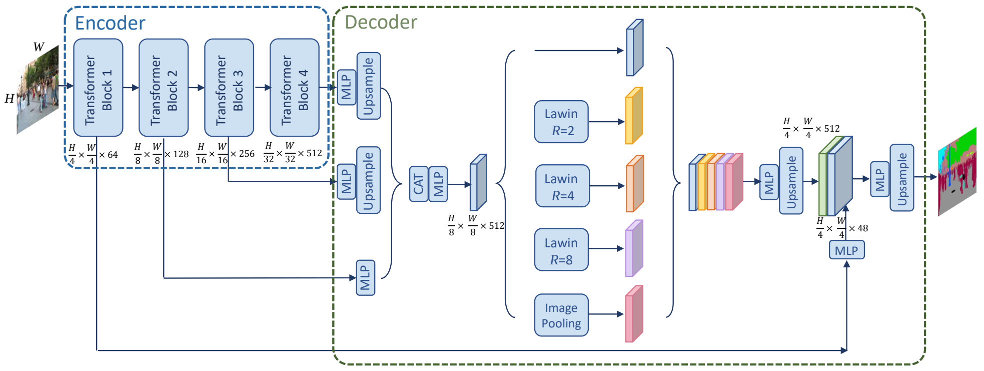


Figure 4: The Lawin architecture

1) Road F1-Score: This metric is the F1-Score computed on the 'Road' class as the number of true positives (TP) is the number of *road* pixels classified as such.

2) Macro F1-Score: Similar to how we compute the Road F1-Score, we can calculate the Non-Road F1-Score. But to have one final value, we need to average the F1-Scores of both classes. The simplest way to do so is to compute the arithmetic mean, and this is directly what the Macro F1-Score does:

$$F1_{\text{macro}} = \frac{F1_{\text{road}} + F1_{\text{background}}}{2}$$

This metric considers all classes of equal importance, which is often desirable.

3) Weighted F1-Score: Note that this is the score used by the Kaggle competition. As its name suggests, the weighted F1-Score weighs the scores of each class by their fraction of presence in the dataset:

$$F1_{\text{weighted}} = \%\text{road} \times F1_{\text{road}} \\ + \%\text{background} \times F1_{\text{background}}$$

A drawback is that contrary to its macro counterpart, the weighted F1-Score can still be very high even if the model performs poorly for a highly underrepresented class.

C. Further Optimization

Here we discuss further optimizations and tricks that empirically improved our submitted results.

1) Transfer Learning: We pre-train on our much larger scraped dataset and then fine-tune on the original dataset, boosting the performance. This technique allows the network to learn richer features while retaining maximal performance on our dataset of interest.

2) Dynamic sample weighting: Similar to AdaBoost [18] but using only one model, we uniformly initialize all training sample weights. The weights are updated during training after each epoch by putting more weight on the samples that incur higher losses, proportionally to $(1 -$ F1-Score$)$. With this technique, the model is encouraged to train more on its weakness, which is especially useful for learning narrower roads that are often not segmented correctly. We then pick the final model from the last training checkpoint, other than creating ensembles with previous checkpoints.

3) Inference-time data augmentation: At inference-time, not only do we feed the test image to the network but also its rotated and flipped versions. For each test image, we ensemble the prediction on itself with its augmented versions. We found this scheme slightly helps to increase the performance as it balances the variance of the test sample, regarding the training data distribution.

4) Different loss functions: We have tried different loss functions to tackle the class imbalance. Namely - due to the nature of some images, roads were not as present as the background was. We have looked into possible solutions, and Focal Tversky loss [19] yields the best performance in the experiments on the U-Net Exp side.

D. Inconclusive experiments

We worked on other approaches without achieving the desired results. Nevertheless, we discuss them here as we believe these are promising ideas and could be further polished in future work:

1) RL based Segmentation (continued): We give more details about the RL-based segmentation framework we briefly introduced in the main part. More generally speaking, the agent learns to paint small areas of the image with a circular brush that can either be put down to paint a road or is variable in width. In our first try, we use a policy network that takes in as input the following observation from the environment, which varies between experiments:

1) A small image patch of the area surrounding the current position of the agent
2) Information on the paint state (paint a road or don't paint a road)
3) The history of the latest past actions
4) A minimap marking the agent's position on the training image as well as its past trajectory

This information is forwarded through convolutional layers for feature extraction before being fed into an MLP that outputs the actions, which vary between experiments:

1) Change in orientation (where to move the agent and if the brush is put down, in which direction to paint the road)
2) Change in magnitude of the movement (if the brush is put down, the magnitude determines the length of the painted road within the seen patch as well)
3) Brush size (as the circle radius)
4) Brush state (put down to paint a road or not)

The input information is supposed to help the agent stay consistent in its predictions over time (e.g., same brush size and walking direction for a straight road and steady road width). It is also enforced to segment and "walk" over the complete image. This is further enforced by the rewards, which contain multiple terms (depending on the experiments):

1) Positively reward the number of correctly classified pixels and penalize wrong predictions
2) Positively reward if new areas are explored (otherwise the agent would stay put)
3) Penalize changes in the brush size if the painting direction does not change and vice

versa (this allows change in road width when taking a turn onto another road, but otherwise enforces same road width and straightness)

4) A small time penalty in order to encourage the agent to segment the image efficiently

Reward functions are notoriously hard to tune in reinforcement learning, and despite many distinct experiments with various parameters, action spaces, and rewards, the network did not learn anything. One reason is that the training takes a long time because the trajectory sampling is inefficient. Hence - we only had a few limited opportunities to adapt the code.

Therefore, we decide to address the high complexity of the reward function by attempting a different approach. We use the fact that the ground truths are readily available to help the network learn a valuable policy. This availability allows us to create a hand-crafted algorithm, which we call the Irresistible Hitchhiker (IR). Given a binary segmentation map, the IR moves along the roads and paints them. While the new network still takes in as input a patch of the neighboring area, the objective is no longer maximizing a reward function that depends on the number of (in)correctly segmented pixels and other parameters but rather is more similar to a regression problem, where the loss metric is the discrepancy between performed movements and those computed by the hand-crafted algorithm mentioned above. Therefore, we train the network to follow the IR's actions.

Despite a great effort, we still did not manage to have the agent learn anything more significant than moving along straight lines. No benchmark for road segmentation in RL to compare our ideas to is a big issue, leaving us only with an inefficient feedback loop made out of our limited tries. Nevertheless, we believe this approach still has potential and could be further explored in future work.

2) Introducing a secondary GAN Loss: When considering the segmentations outputted by some model, it is almost always the case that a human can surely tell it apart from a hand-made segmentation. Indeed, automatic segmentations are usually characterized by the presence of spurious blobs, as well as roads that are either disconnected or with varying widths. The idea is to introduce a discriminator model jointly learned with the segmentation model that distinguishes between generated segmentations (fake) and real ones. The GAN loss is then added to the base segmentation loss (dice, BCE, or any other loss) as a measure of the cleanliness of the segmentation. In practice, experiments with this scheme did not manage to improve the base segmentations, most probably due to training instability.

3) Incremental image segmentation: The idea is to have an image through a first network that outputs a segmentation. Then, we feed it together with the original image through the (same) network. The hope was for the preliminary segmentation to act similarly to an attention channel (the network would learn to focus more on the parts segmented in the previous pass and hence output a better final segmentation). While this scheme maintains the same number of parameters, in practice, it probably makes the optimization problem much harder, which makes it harder for the network to converge towards a favorable configuration.

4) AdaBoost: We adapt the classical AdaBoost algorithm to the task of image segmentation. Before the first training, we initialize all samples with the same weight in a random dataset sampler. After a model has finished training, we use the road F1-score of each training sample as a weight for the next training. This approach is different from the binary classification in the original algorithm. With the road F1-score, we hope to inhibit the model's tendency to ignore small roads and predict false negatives. Furthermore, we determine the model's weight by the F1 score on the validation dataset as a measure for generalization on unseen data. We try AdaBoost by ensembling ten models but find no improvements regarding the F1-score. A possible explanation is that the models get biased towards the initially bad weights and overfit on them. A possible solution is to decrease the impact of the current performance on the sample weight more than in AdaBoost, only changing it very slightly. Furthermore, ensembling many more models could also improve the result, which was no tested due to lack of time.

5) MonoBoost: As opposed to AdaBoost, which trains multiple models, this new variant only trains a single one, as the name suggests. The motivation behind this is simple: AdaBoost typically trains weak (and hence fast) learners, whereas we are trying to apply a similar technique on top of strong models that require a long training time. At each iteration, instead of training a new model from scratch, MonoBoost continues training the model from the previous iteration but with different sample weights. We perform the recomputation of weights as follows: each training sample passes through ResNet, known as a feature extractor, where the output at layer $7c$ is selected. For the ith sample, let's call the outputted feature vector $f_{i,\text{train}}$. We do the same for each validation sample j to obtain the feature vectors $f_{j,\text{val}}$. From these vectors, we can compute a similarity phi_{ij} between training sample i and validation sample j as:

$$\phi_{ij} = e^{-d_{ij}} \in [0,1], \quad d_{ij} = L2(f_{i,\text{train}} - f_{j,\text{val}})$$

with $L2$ denoting the L2-norm. At the end of each iteration, we compute the error e_j for each validation sample j using the weighted F1-score as:

$$e_j = 1 - \text{"weighted F1-score}(j)\text{"}$$

The new weight for sample i is finally given by:

$$w_i = \frac{1}{Z} \sum_j \phi_{ij} e_j,$$

where Z is a normalization constant:

$$Z = \sum_{i,j} \phi_{ij} e_j$$

that makes sure $\sum_i w_i = 1$. Since we only have a single final model, the averaging performed by AdaBoost is implicit, and we only need to retain the last checkpoint obtained after a fixed number of iterations. Unfortunately, though this scheme did not deteriorate our results, it also did not manage to improve them. A possible next step would be to create our feature extractor specifically trained on our segmentation dataset instead of the generic ResNet, which would be able to yield more meaningful similarities between the training and the validation samples.

E. The scraped dataset

Though not perfectly similar to the original data, our scraped dataset is still relatively close to Boston and Los Angeles roads given in the original dataset. We depict the comparison between samples of both datasets in Figure 5.

F. Model size comparison

	Number of parameters
U-Net3+	16.13M
U-Net3+ (+VGG)	31.92M
DeepLabV3	39.64M
SegFormer	84.7M
Lawin	86.9M
U-Net Exp	24.48M
U-Net Exp (+ VGG)	48.30M

Table II: The number of parameters used by each network. (+VGG) indicates the size of the model with a VGG backbone.

G. UNet vs. UNet3+ vs. U-Net Exp

We visualize side-by-side the original UNet, the UNet3+, as well as our U-Net Exp architecture in Figure 6

H. Example segmentations

Figure 7 shows the segmentations predicted by a Lawin [14] model on the 25 images of our validation set, overlayed on top of the satellite views.

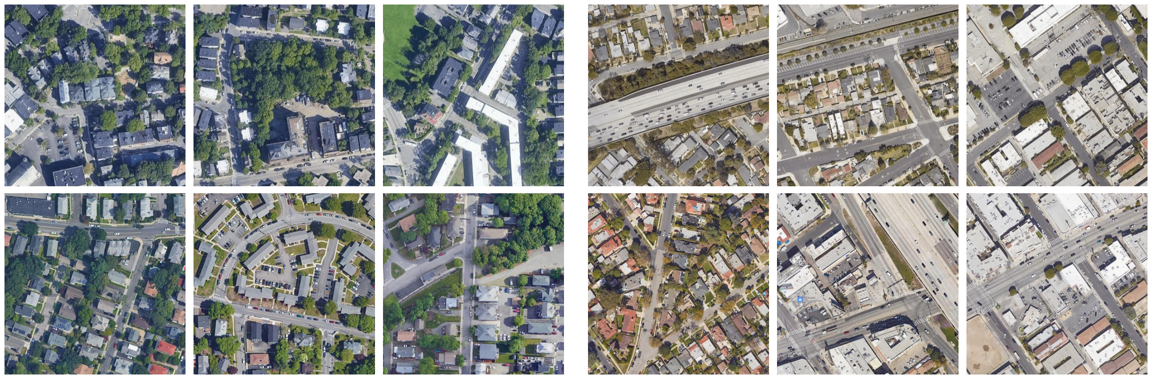


Figure 5: Comparison between samples from the original dataset (top) and our scraped dataset (bottom)

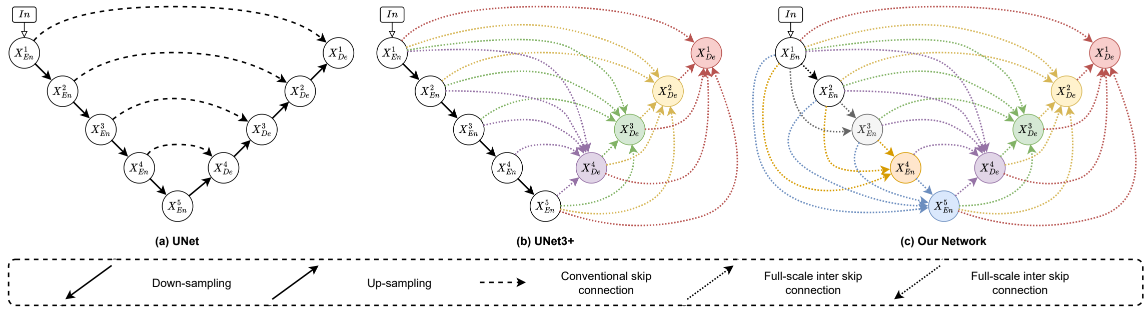

(a) UNet
(b) UNet3+
(c) Our Network
Down-sampling
Up-sampling
Conventional skip connection
Full-scale inter skip connection
Full-scale inter skip connection

Figure 6: Comparison between UNet, UNet3+ and our architecture

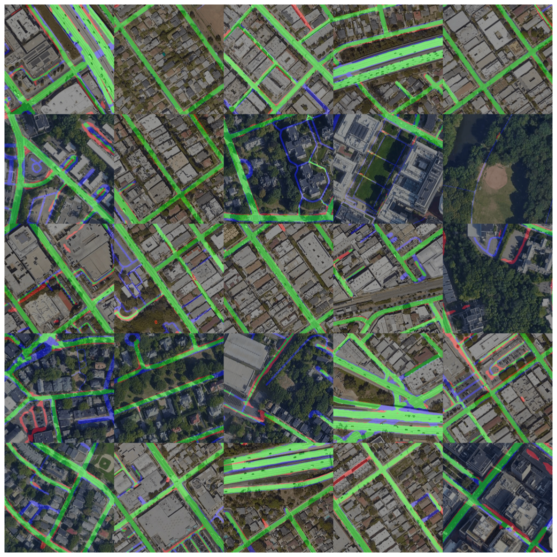


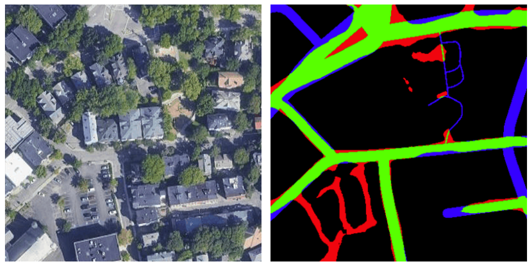
Figure 7: Example segmentation given by the Lawin network. True positives are shown in green, false negatives in blue and false positives in red.

I. Erroneous ground truths

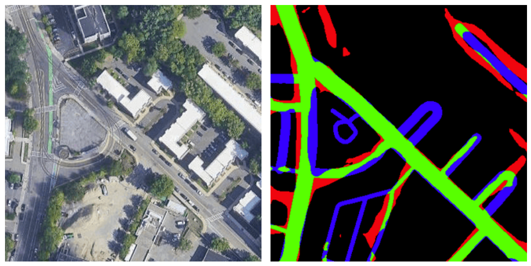
We notice that the trained models sometimes output more correct segmentations than the given ground truth masks, which get classified as errors. While this sets a plateau on the maximum achievable F1-Score, it suggests that neural networks can guide humans and correct their mistakes. We demonstrate two such examples in Figure 8.

J. Impact of transfer learning

The U-Net3+ as well as our U-Net Exp use the VGG-19 backbone whereas the DeepLabV3, the SegFormer and Lawin use the ResNet-50 backbone:



(a) The network correctly classifies the bottom-left part of the image as being part of a road, yet the ground truth disagrees.



(b) The network correctly classifies the top-right corner of the image as a road, whereas the ground truth classifies the trees as roads.

Figure 8: Ground truth anomalies. True positives are shown in green, false negatives in blue and false positives in red.

	Metrics		
	Road F1	Macro F1	Weighted F1
ResNet-50			
DeepLabV3	0.738	0.845	0.923
SegFormer	0.766	0.861	0.931
Lawin	0.772	0.865	0.932
VGG-19			
U-Net3+	0.707	0.826	0.914
U-Net Exp	0.712	0.830	0.916

Table III: Performance of the baselines and our model evaluated on the competition dataset, using transfer learning.