CIL Project Report – Road Segmentation

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

***Abstract*—1) State the problem, 2) State why it is an interesting problem, 3) Say what your solution achieves, 4) Say what follows from your solution**
We should only write the abstract at the end

I. Introduction

Binary Image segmentation is a per-pixel binary classification task. Hence, segmenting an image amounts to performing a classification task with an exponential number of data points, where each possible segmentation represents a new result. An important insight that makes this problem tractable is that the pixels are not independent but correlated, as images have structure. Models that can capture these correlations are naturally well-suited for such a task and in particular, CNNs are the most well-known and successful models in this domain. One of the first architectures specifically tailored toward image segmentation was the pyramidically-structured UNet [1], which achieved state-of-the-art medical results in 2016. Since then, a plethora of related models have been developed, such as, UNet++ [2], Attention UNet [3], U^2-Net [4], ResUNet [5] etc.

Maybe move part of this paragraph to the abstract? There are numerous areas where image segmentation can be beneficial, but in this paper - we are focusing on the road segmentation task. Indeed, being at the center of human mobility, communication, and ultimately civilization, an enormous number of kilometers of roads have been built around the globe, that manually segmenting them has become an almost impossible task. However, with the increasing availability of satellite images, it has fortunately become possible to tackle such a task using Machine Learning tools. We propose a new architecture called UNetExp that builds on top of U-Net3+ [6], surpassing its performance. We also use techniques such as: boosting, ensembling, and transfer learning, which enormously improved the performance of our models.

The structure for the remainder of this report is as follows: In Section II, we describe how we extended the original available data both by acquiring new images and augmenting existing ones. Section III outlines the baselines 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 present in the Appendix.

II. Extending the dataset

The original dataset only 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 (note that wee keep the original resolution of 400x400 for all images):

A. Augmentation of the old data

We augment each image by applying geometric transformations such as horizontal and vertical flips, rotations, translations, and scaling, as well as lighting transformations - changing the contrast, luminosity, and saturation. This simple approach allows us to produce reasonable images for our task while increasing the performance of our models. Two modes were tested, with varying results. The first technique created new data through the mentioned augmentations and saved them. Then, we used these new and old images to generate predictions. The second mode created these new images on-the-fly by loading an image and perturbing it randomly before using it for predictions. ***We

should quantify this***. However, we can only get so much variation from these transformations, with diminishing returns, the more augmentations we add.

B. Obtaining new data

The second approach focused on acquiring new data. The biggest challenge was to get images comparable in distribution to the original one so as not to skew the total dataset's distribution in the direction of the newly obtained data. A skew like this would cause the model to perform poorly on the test images from the original distribution. We carefully went through the original dataset, investigating the given data, and concluded that it consists of images retrieved from two cities in the US - Boston and Los Angeles. We then automatically scraped additional data ***how many in total*** from these regions and hand-filtered them in order only to keep those that we considered belonged to the original distribution. We visualized some samples from the extracted dataset in the APPENDIX.

III. MODELS AND METHODS

For this project, we built a cross-platform (Windows/Linux) framework that handles both PyTorch and TensorFlow models, with an MLFlow backend to store, share and monitor our experiments simultaneously. Moreover, this framework helped us get better insights about our models by logging all necessary details: code-base snapshots, command-line arguments, hyper-parameters used, checkpoints, metrics, etc.

A. Baselines

To evaluate our models, we compare them to 5 baselines, namely:

1) **U-Net [1]:** The plain original U-Net architecture
2) **U-Net3+ [6]:** The U-Net variant on top of which our U-Net Exp is built
3) **DeepLabV3 [7]:** A strong and very common baseline
4) **SegFormer [8]:** One of the most powerful segmentation networks
5) **Lawin [9]:** LaWin for the Win

Here we should mention how we made sure that all the runs are comparable, and overall give more detail as to how to reproduce our results.

Performance of these networks and comparison to our work is shown in section IV.

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

B. UNet Exp

We create UNet Exp based on the architecture of UNet3+ [6]. The architecture of UNet3+ extends UNet's architecture by proposing new inter-skip connections that propagate the activation signal from one part of the network to the other, changing the old skip connections. UNet3+ suggested extensions were only present in the decoder part. However, we found in our experiments that the creation of similar links within the encoder structure is beneficial. We believe we can make more accurate predictions without extending the decoder by creating better representation within the encoder.

The new inter-skip connection gathers information from past layers and convolves each layer's result to assemble it into new input to the existing UNet nodes. 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. Each inter-skip connection can be represented by:

$$X_{De}^{i} = \begin{cases} X_{En}^{i} & \text{, if } i = N \\ C\{C(D(X_{En}^{k}))_{k=1}^{i-1}, C(X_{De}^{i}, C(U(X_{De}^{k}))_{k=i+1}^{N}\} & \text{, otherwise,} \end{cases} \quad (1)$$

Figure 1: Encoder 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 each. 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.

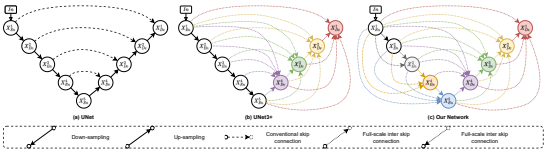


Figure 2: Comparison between UNet, UNet3+ and our architecture

In our architecture, we perform similar steps on the encoder side of the network (see figure 3). 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, with the number of filters adjusted by the previous block's size, to allow 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.

$$X_{De}^{i} = \begin{cases} C(In) & \text{, if } i = 1 \\ C\{C(D(X_{De}^{k}))_{k=1}^{i-1}\} & \text{, otherwise,} \end{cases} \quad (2)$$

Figure 3: Decoder inter-skip connection

Moreover, we changed the ReLU activation function to the SiLU [10] activation function (see figure 4) within our architecture, as we believe the SiLU activation is more suited for the task at hand.

$$f(x) = \frac{x}{1 + e^{-x}}$$

Figure 4: SiLU [10] activation function

SiLU alleviates the vanishing gradient problem, which might 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 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 was also augmented and tested with different pre-trained models. We observed the most considerable performance improvement with the VGG-19 [11] network treated as a backbone, which was mixed into the decoder using our inter-skip connections. It provided a better initial gradient and stabilized the learning.

C. Model Selection

We perform hyper-parameter searches on our various models using the HyperOpt [12] framework, which uses heuristics to optimize the search speed. In particular, we optimize our models for the segmentation threshold both at train-time 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 are deep and contain parameters in order of tens of millions, they can easily overfit the limited data if one is not careful. To alleviate this problem, we perform 3-fold cross-validation during hyper-parameter tuning. After the hyper-parameters selection, we train for one last time on the whole dataset (without a validation split) for the final model to not waste any data.

D. Ensembling

To further improve our score, we use ensembling techniques to merge the predictions of our well-performing networks. We tried two different approaches. Our first method consisted of a simple ensembling of the rounded classifications of each network, performing per-pixel majority voting. By interpreting the per-pixel output of a network as binary class probabilities, averaging the outputs of

all networks in the ensemble before rounding to either class allowed us to take into account the predictions of individual networks and measure their confidence, leading to an even higher performance boost compared to our first approach.

E. Transfer Learning

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

TODO quantify the performance difference.

F. Further optimizations

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

- **Sample weighting:** Similar to AdaBoost, we uniformly initialize all sample weights and update them during training by putting more weight on the samples that incur higher losses, where the sample weight is proportional to $(1 - \text{F1-Score})$.
- **Inference-time augmentation:** At inference-time, we not only feed the test image to the network but also rotated/flipped versions and ensemble the prediction outputted for each sample. We found this scheme slightly helps increase performance.
- **AdaBoost** TODO: move this section to the appendix if it does not improve the results or leave it here otherwise
- **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 standard binary cross-entropy, Dice Loss, based on the Dice Coefficient, and Focal losses - Focal loss and Focal Tversky loss. In the experiments on the UNetExp side, Focal Tversky loss seemed to yield the best performance.
- more..?

IV. Results

The performance of the UNetExp network as compared to our baselines is shown in Table I.

	Metrics		
	F1-Score	Weighted F1	Others?
U-Net	-	-	-
U-Net3+	-	-	-
DeepLabV3	-	-	-
SegFormer	-	-	-
Lawin	-	-	-
UNetExp (Ours)	-	-	-

Table I: Baselines performance evaluated on the competition dataset

As we can see, blablabla

V. Discussion

bla bla bla

We also experimented with many other different approaches but did not get any significant improvements. Due to lack of space, we document them in the Appendix.

An interesting thing we noticed is that the trained models sometimes output more correct segmentations than the ground-truth mask is, which gets 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 depict the observed phenomenon in the Appendix section.

VI. Summary

In this project, we developed a significant ML framework and extended existing architectures by creating our own U-Net version, the UNetExp, which performs significantly better than its baselines. We also came up with the idea of dynamically varying segmentation thresholds, which can potentially boost the performance of other networks for other tasks. Furthermore, we... [summarize the techniques we used]

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] 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

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

[8] 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

[9] 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

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

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

[12] 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

APPENDIX

A. Baseline architectures

1) **U-Net:** This novel encoder-decoder architecture was a breakthrough in image segmentation thanks to its skip connections allowing the decoder part of the network to preserve spatial information, correctly positioning the extracted feature maps during reconstruction.
2) **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 (Refer to Figure 6). 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.
3) **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), helping broaden the field of view of the network, allowing to capture long-range context.
4) **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.

5) **Lawin:** This architecture shares some similarities with the SegFormer as it is also a Transformer and employs MLPs. Similar to DeepLabV3, the Lawin network uses a Spatial Pyramid Pooling (SPP) scheme but replaces the Atrous convolutions by what's called a Large Window Attention, yielding the LawinSPP, which acts as the decoder part of the network.

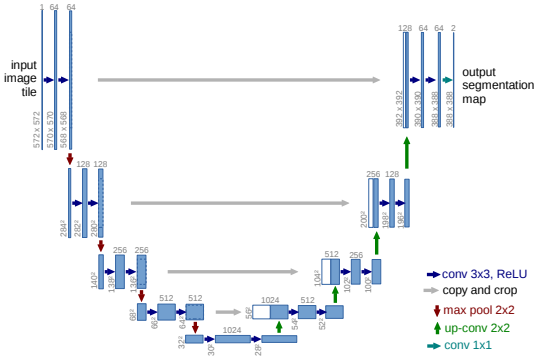


Figure 5: The U-Net architecture

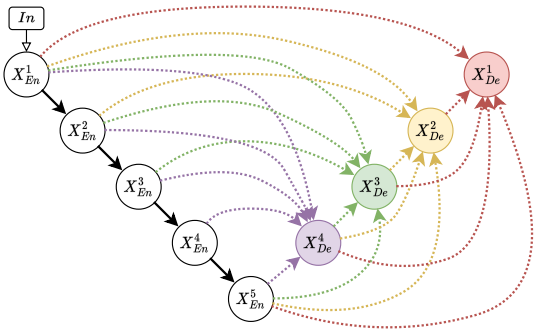


Figure 6: The U-Net3+ architecture

B. The scraped dataset

Though not perfectly similar to the original data, our scraped dataset is still very close, both for Boston and Los Angeles roads.

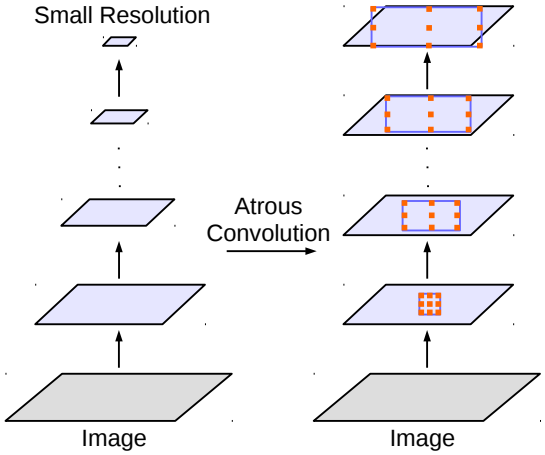


Figure 7: Illustration of the DeepLabV3's atrous convolution, allowing to preserve larger resolution

C. Example segmentations

Figure 11 shows the segmentations predicted by ¡insert_mode_name¿ on the 25 images of our validation set, overlayed on top of the satellite views.

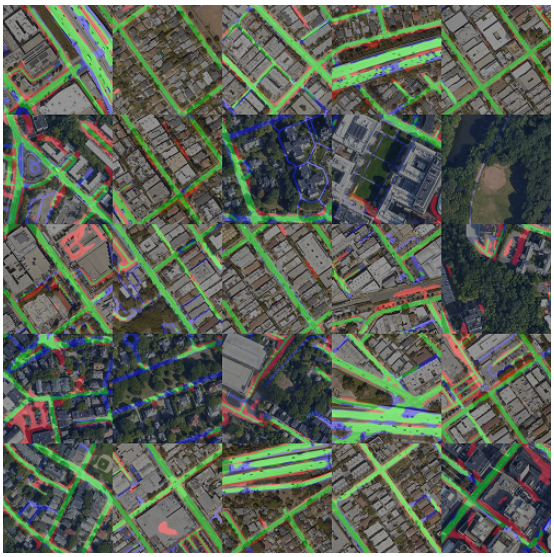


Figure 11: Segmentations obtained using our final model. True positives are shown in green, false negatives in blue and false positives in red.

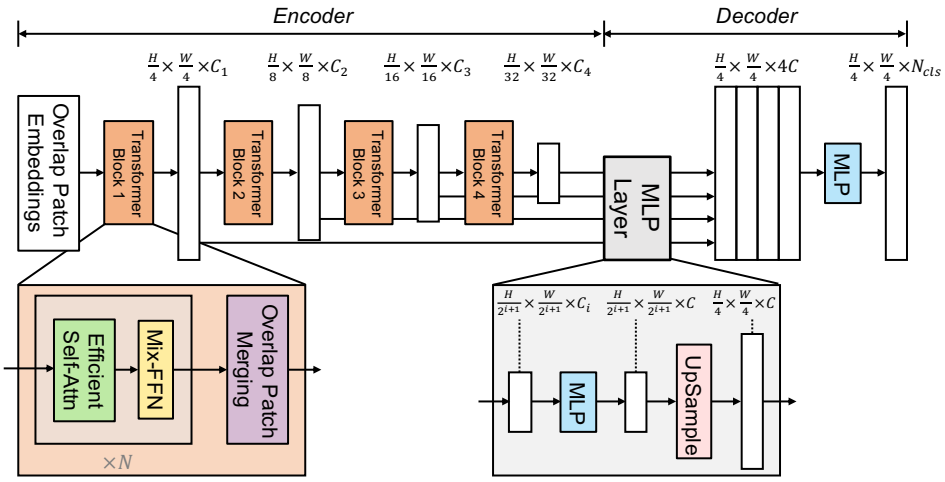


Figure 8: The SegFormer architecture

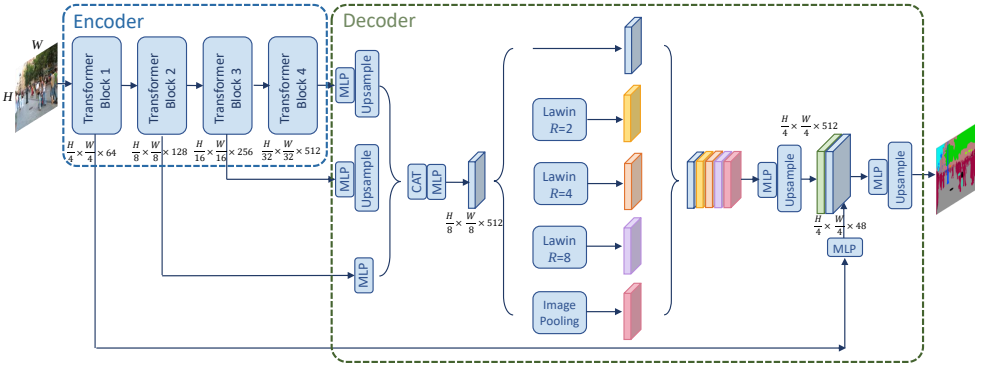


Figure 9: The Lawin architecture

D. Inconclusive experiments

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

1) Reinforcement Learning based Segmentation: Current segmentation networks ultimately perform per-pixel classification, which tends to create artifacts, where some parts of the segmentation end up disconnected from the others, or contain jittery boundaries. Based on this observation, we attempted a novel Reinforcement Learning architecture with built-in segmentation 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 on its current position and if yes, the appropriate "brush" size. On our first iteration, we used a policy network that takes in as input a small patch of the area surrounding the current position of the agent that is fed through convolutional layers for feature extraction, before being fed into an MLP that outputs the actions (change in orientation,

Figure 10: Comparison between samples from the original dataset (top) and our scraped dataset (bottom)

brush size etc). The reward contained multiple terms. A term that positively rewarded the number of correctly classified pixels and vice-versa, a term that encouraged exploration (otherwise the agent would get scared and stay put), a term that penalized sudden direction changes (roads are typically straight) etc. Reward functions are notoriously hard to tune in reinforcement learning and despite many different experiments with various parameters, the network was not able to learn anything.

We decided to address the high complexity of the reward function by attempting a different approach: to help the network learn a useful policy, we made use of the fact that the groundtruths were readily available. This allowed us to create a hand-crafted algorithm **(Name of the algorithm?)** that given a binary segmentation map, 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 based on maximizing a reward function that depends on the number of (in)correctly segmented pixels and other parameters, but rather is seen as a regression problem where the loss metric is based on the discrepancy between taken actions and those computed by the hand-crafted algorithm mentioned above.

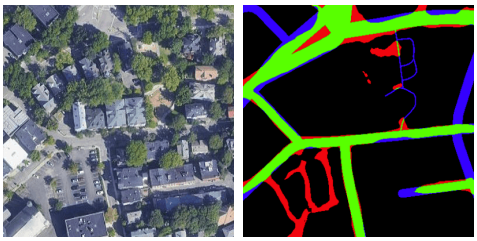
Despite a great effort, we still did not manage to have the network/agent learn anything more significant than moving along straight lines. Nevertheless, we believe this approach still has a lot of potential and could be further explored in future work.

2) Introducing a secondary GAN Loss: When considering the segmentations outputted by some model, it is almost always the case that a human can easily 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 width. The idea is then 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 cleanliness of the segmentation. In practice, experiments with this scheme did not manage to improve the base segmentations, most probably due to training unstability.

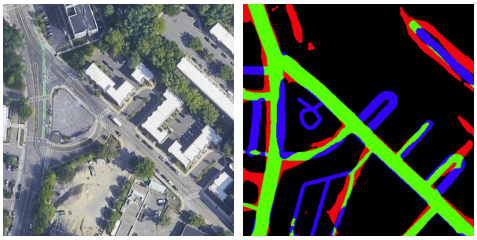
3) Incremental image segmentation: The idea is to have an image through a first network that output a segmentation, then feed it together with the original image through the (same) network. The hope was for the preliminary segmentation to act as some sort of attention channel (the network would learn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

E. Erroneous groundtruths

As discussed in the main part of the paper, some segmentations are impossible to get 'right' other than by completely overfitting to the data (which we obviously don't want to), due to the groundtruth segmentations being wrong. Two such examples are shown in Figure 12.



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



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

Figure 12: Groundtruth anomalies