Single-Shot End-to-end Road Graph Extraction
Supplementary Material

Gaetan Bahl[1,3] Mehdi Bahri[2] Florent Lafarge[3]
[1]IRT Saint Exupéry [2]Imperial College London [3]Université Côte d'Azur - Inria
{gaetan.bahl,florent.lafarge}@inria.fr, m.bahri@imperial.ac.uk

1. Inference on large images

We found that the output of the link prediction branch is tied to the training image size. Thus, we did not obtain good results by directly inferring on larger images when the network was trained on small images. To circumvent this issue, we apply our network on the large images of the RoadTracer test set using a sliding window of the same size as the training patches, and accumulate the detected edges.

2. Impact of choosing a smaller backbone

Feature extractors (backbones) come in different sizes to suit different kinds of performance targets. For example, some applications like edge computing might accept a reduction on accuracy in favor of faster inference time and higher throughput. In order to evaluate the impact of choosing a smaller backbone on our method, we take the network presented in the main text and replace the ResNet-50 backbone with ResNet-18.

Method	APLS	Single image time (s)
Ours (R18)	38.80	**19.6**
Ours (R50)	**40.45**	20.9

Table 1. Run-time in seconds on the RoadTracer Dataset. The single image score is averaged over the whole test set. Using a smaller backbone (R18) is slightly faster but causes a small drop in APLS.

The results of this experiment are presented in Table 1. We observe that using a ResNet-18 backbone is slightly faster but results in a slightly lower APLS score. This means that our method works with smaller backbones and thus can be used on a variety of low power devices, especially ones that have a low amount of memory, such as the Nvidia Jetson family of devices.

3. Impact of edge detection threshold

In this section, we study the impact of the edge detection threshold, which can be freely chosen between 0 and 1. To do this, we compute the three main accuracy metrics for a range of thresholds varying between 0 and 0.5, using our main model. Setting a high detection threshold favors the Precision of edges, while setting a low threshold favors the Recall. Figure 1 shows the results of this experiment. We observe that the P-F1 (pixel metric) and APLS are better when detecting more edges, while the J-F1 (junction metric) is slightly better when detecting fewer edges. This makes sense as the J-F1 is based on having the correct number of edges for each junction, while the APLS should benefit from more path options. The P-F1 may be higher at low thresholds simply because we find more road pixels. This experiment shows that finding and setting an appropriate edge threshold for each model is important in order to obtain the best accuracy and the best compromise between the three metrics.

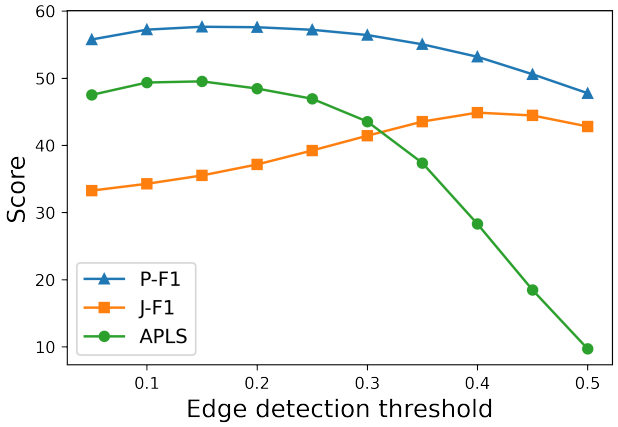

Figure 1. J-F1, P-F1 and APLS scores for our main model, at a range of edge detection thresholds.

4. On the J-F1 metric

As previously said in the Method and Limitations sections of our main text, our method is only able to find a single point-of-interest per output cell, which can lead to

merged junctions in the output graph. In addition, as seen in the Experiments section of the main text, our method will inherently find a lower number of junctions since it is tailored towards the extraction of a sparse graph. Since the J-F1 score is based on a matching of junctions within a certain radius, we believe that these design choices significantly affect this metric, which leads to our method scoring lower than some other approaches.

5. GNN model choices and ablation study

In this section, we perform a comparative study of different design decisions for the GNN portion of the model.

We compare the following choices for the construction of the road graph:

1. Using the complete graph as the supporting graph for each GNN layer and treating the problem as a pure edge classification task
2. Initializing the road graph by k-NN search (k = 4) on the output of the node feature branch (dim = 256) to which we concatenate the Cartesian coordinates of the junctions (dim = 2)
3. Dynamic construction of the graph at each layer by k-NN search (k = 4) on the layer's input (*i.e.*, as described in point 2 for the first layer, and on the output features of the previous layer for subsequent layers)

We also compare two choices of edge scoring function:

1. A 2-layer MLP (FC(256) $\rightarrow$ FC(256) where FC(N) denotes a fully-connected layer with N output features)
2. A $256 \times 256 \times 1$ bilinear layer

We use a three-layer graph neural network based on the EdgeConv operator with BatchNormalization and ReLU activations. We parameterize the EdgeConv operators each with a linear layer with 256 output features. We also report the results of baseline models trained without graph convolutions.

Finally, we compare the 3-layer EdgeConv networks with DeepGCNs [6, 7] using the Generalized Graph Convolution (GENConv) operator [7], layer normalization [2] and ReLU activations. We apply the DeepGCN layers sequentially on:

1. The raw image features produced by the backbone (DeepGCN)
2. The output of one convolutional layer in the node feature branch (1 Conv + DeepGCN)
3. The output of 4 convolutional layers in the node feature branch (4 Conv + DeepGCN) as for the EdgeConv-based models

In case (1) we used 7 layers of (GENConv $\rightarrow$ LayerNorm $\rightarrow$ ReLU), in case (2) we used 6 layers, and in case (3) we used 3 layers, so as to keep model capacity comparable with the EdgeConv models applied on the full node feature branch. We applied the DeepGCNs on the complete graphs only.

For each of these experiments, we report the J-F1, P-F1 and APLS metrics at the edge threshold that maximizes their sum. Table 2 shows the results of these experiments. To enable faster experimentation, we initialized some of the models using the pre-trained weights of a "Baseline MLP" model trained on the complete graph, such models are denoted by a checkmark in the "Pre-trained" columns of Table 2.

We can draw the following conclusions from the experimental results shown:

- The baseline with the bilinear classifier outperforms the baseline with a 2-layer MLP, which is expected since it has a much larger number of parameters. While the improvement is noticeable, the model loses compactness and efficiency.
- All three constructions of the graph (complete, static k-NN and fully dynamic) are able to perform well and to outperform the MLP and the Bilinear baseline. Notably, the models that combine a GNN with an MLP classifier outperformed the ones with the same GNNs but a bilinear classifier (scoring function). The entire GNN adds fewer parameter than changing the MLP for a bilinear layer, and yet can bring larger performance deltas, which further motivates our choice of using an MLP scoring function.
- The best performing GNN with the EdgeConv operator used a fixed 4-NN graph support for message passing (but still scored all possible edges for the final graph). However, choosing the right value of k adds another element to hyperparameter tuning of the model and comes at the disadvantage of reduced robustness to cases where images have no junctions to detect (*e.g.* satellite images of large bodies of water such as lakes). We therefore chose to report the performance of the simpler model in the main text, while showcasing that sparse graph priors may indeed lead to better road graph reconstructions.
- Getting rid of the node features branch reduces the number of trainable parameters but appears to be detrimental to the performance in most cases. Experiments with deeper GNNs applied directly on the output of the backbone showed increased performance compared to shallower GNNs, although the graph construction differs (l. 14, 17)

#	Pre-trained	Init Graph	GNN	Classifier	Raw feat	Wait	Train j_{thr}	J-F1	P-F1	APLS	Sum
1	✓	Baseline	✗	Bilinear				**40.33**	55.93	43.95	140.21
2		Baseline	✗	Bilinear				36.03	53.87	41.64	131.54
3		Baseline	✗	Bilin small				32.24	50.4	36.11	118.75
4	✓	Baseline	✗	MLP				36.92	53.43	41.17	131.53
5		Baseline	✗	MLP				35.46	56.03	45.35	136.84
6	✓	Complete	EdgeConv	Bilinear				37.69	56.04	45.39	139.13
7*	✓	Complete	EdgeConv	MLP				39.23	57.2	46.93	143.36
8	✓	Dynamic	EdgeConv	Bilinear				35.63	56.29	46.04	137.96
9	✓	Dynamic	EdgeConv	Bilinear			0.3	37.90	55.5	43.68	137.08
10	✓	Dynamic	EdgeConv	Bilinear	✓			37.32	52.07	38.51	127.91
11	✓	Dynamic	EdgeConv	Bilinear	✓		0.3	37.92	52.37	40.27	130.55
12	✓	Dynamic	EdgeConv	MLP				38.44	56.7	46.21	141.35
13	✓	Dynamic	EdgeConv	MLP	✓			34.95	53.75	41.57	130.27
14	✓	Dynamic	EdgeConv	MLP	✓		0.3	35.54	56.5	45.91	137.95
15	✓	kNN-4	EdgeConv	Bilinear				37.30	54.25	42.28	133.83
16	✓	kNN-4	EdgeConv	MLP				37.41	**57.54**	**48.71**	**143.66**
17	✓	Complete	DeepGCN	MLP	✓			37.17	55.43	42.59	135.19
18	✓	Complete	1 Conv + DeepGCN	MLP				38.35	57.11	47.12	142.58
19	✓	Complete	4 Conv + DeepGCN	MLP				37.89	**57.71**	**48.84**	**144.44**
20		Dynamic	EdgeConv	Bilinear				34.52	50.74	35.52	120.78
21		Dynamic	EdgeConv	Bilinear	✓	30		33.23	46.26	30.71	110.20
22		Dynamic	EdgeConv	Bilinear		30		32.03	50.94	37.31	120.28
23		Dynamic	EdgeConv	Bilinear	✓	30	0.3	36.89	49.02	32.13	118.04
24		Dynamic	EdgeConv	MLP				36.35	56.31	45.55	138.21
25		Dynamic	EdgeConv	MLP		30		36.54	55.72	43.97	136.22
26		Dynamic	EdgeConv	MLP		30	0.3	34.32	55.41	45.65	135.38
27		Dynamic	EdgeConv	MLP		10	0.3	35.36	54.96	44.44	134.76
28		Dynamic	EdgeConv	MLP	✓	30	0.3	34.48	55.02	42.15	131.65
29		Dynamic	EdgeConv	MLP	✓	30		35.73	54.9	41.91	132.54
30		Dynamic	EdgeConv	MLP	✓	10	0.3	34.90	56.03	44.60	135.54

Table 2. Variations on our model. Pre-trained = use of pre-trained ResNet and junctionness/offset branches for faster training. Raw feat = GNN applied directly to ResNet features (no node feature convolutions). Wait = Number of epochs where only the junction-ness and offset branches are trained before training the edge branch (default: 0). Train j_{thr} = junction-ness threshold used during training (default: 0.5). Sum = J-F1 + P-F1 + APLS. *variation presented in the main text

- Compared to 3-layer EdgeConv networks on the complete graphs, 3-layer DeepGCNs applied following either a 1-layer or 4-layer node feature branch performed better. These models also outperformed the dynamic graph models and the 3-layer EdgeConv applied on a sparse 4-NN graph (by a slimmer margin in the latter case). The DeepGCNs applied on raw features outperformed some of the shallower models trained with a node feature branch, but did not match the best performing models.

- All DeepGCNs outperformed the MLP baseline, which indicates that, keeping the edge scoring function the same, replacing all (DeepGCN, l. 17)or most (1 Conv + DeepGCN, l. 18)2D convolutions with graph convolutions leads to increased performance compared to a model that only uses 2D convolutions.

Deep Graph Convolutional Networks We suspect the last two points are due to two effects: first, 2D convolutions can be seen as special cases of graph convolutions applied to the 2D lattice graph while enforcing translation equivariance; their inductive bias is well suited to the processing of images. In contrast, we applied the GNNs (EdgeConv or DeepGCN) on the (dynamic or complete) graphs of detected junctions, which means the GNNs do not have access to the neighboring pixel's context whereas the 2D Euclidean convolutions do. We believe this contributes to the reduced performance of all graph-convolutional models compared to models that combine 2D convolutions and graph convolutions. Additionally, the higher performance of the graph convolution models compared to only using 2D convolu-

tions - especially on the APLS metric - show the contribution of the graph-based models.

Regarding the relative performance of DeepGCNs compared to shallower EdgeConv models, the graphs extracted from the images are small: they have at most $16 \times 16 = 256$ nodes and $(256) * (256 - 1)/2 = 32640$ edges. The increase in receptive field that comes with deeper models will lose effectiveness once the entire graph is covered at a given layer. Furthermore, the experiments we report with DeepGCNs in Table 2 are done using the complete graph as support. Messages from each node reach all other nodes in a single iteration, which reduces the benefits one can glean from using deeper models, and makes the GNNs more susceptible to the smoothing problem [9]. We therefore decided to report results using shallow GNNs using the EdgeConv operator which is well suited to learning edge features and to learning on dynamic graphs. Further work will investigate using DeepGCNs on sparse graphs as well as on graphs built on larger images.

Junctionness threshold In addition to the ablation study, we evaluated the impact of the junction-ness threshold j_{thr} used during training. Lowering this threshold seems to have a positive impact on the three metrics. Our intuition is that when the junction-ness threshold is lower, the subsequent GNN is presented with more nodes and a larger number of possible edges for each image, and is thus trained better and faster.

6. Other uses of our method

Our method is generic in the sense that it could be used for applications other than road extraction. In fact, it could be suitable for any image to 2D graph application. For example, our method could be used for blood vessel extraction as done in [13]. Another example is the polygonal extraction of buildings in aerial images. For this task, just like for road extraction, existing methods either rely on an iterative process [10] or on post-processing of a pixel-based segmentation [8]. Thus, for this task, our method is able to provide the same advantages as for road extraction. Figure 2 shows an example of building extraction using our method.

7. Qualitative results

Figures 3 and 4 show more qualitative results on cities from the RoadTracer test set. Our method is able to find more roads than RoadTracer [3] and DeepRoadMapper [11]. Some roads and highways that cannot be accessed easily by iterative approaches are found by our method. Our graphs also seem to have a better connectivity than the ones found by the segmentation-based method.

Figure 2. Our method can be used for other tasks, such as building extraction. This example is taken from the CrowdAI Mapping Challenge dataset [12]

8. Environmental impact statement

While the focus of our work is to create efficient neural networks that are able to run on very low power devices, we cannot help but notice that these networks are still created and trained using power-hungry multi-GPU machines and wonder about the environmental impact. For this paper, we tracked the number of GPU hours used and use that to estimate its global environmental footprint and publish these estimations, as recommended in [1, 4, 5].

In order to run the experiments required for our main results and ablation study, we have used 1572 GPU hours on Nvidia Tesla V100 GPUs, which are rated for a power consumption of 300W. This, not counting CPUs, cooling, PSU efficiency, storage of datasets and results, as well as different trials or hyper-parameter searches on workstations, amounts to 471,6kWh. Since the carbon intensity of our electricity grid is 10 gCO2/kWh, we estimate an emission of 4716 gCO2, which is equivalent to 39.17 km traveled by car according to [1].

References

[1] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv:2007.03051*, 2020. 4

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2

[3] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *CVPR*, 2018. 4, 6, 7

[4] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248), 2020. 4

[5] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: Quantifying the carbon emissions of computation. *arXiv:2007.07610*, 2020. 4

[6] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019. 2

[7] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020. 2

[8] Muxingzi Li, Florent Lafarge, and Renaud Marlet. Approximating shapes in images with low-complexity polygons. In *CVPR*, 2020. 4

[9] Qimai Li, Zhichao Han, and Xiao Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018. 4

[10] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Topological map extraction from overhead images. In *ICCV*, 2019. 4

[11] Gellert Mattyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. *ICCV*, 2017. 4, 6, 7

[12] Sharada Prasanna Mohanty, Jakub Czakon, Kamil A Kaczmarek, Andrzej Pyskir, Piotr Tarasiewicz, Saket Kunwar, Janick Rohrbach, Dave Luo, Manjunath Prasad, Sascha Fleer, et al. Deep learning for understanding satellite imagery: An experimental survey. *Frontiers in Artificial Intelligence*, 3, 2020. 4

[13] Carles Ventura, Jordi Pont-Tuset, Sergi Caelles, Kevis Kokitsi Maninis, and Luc Van Gool. Iterative deep learning for road topology extraction. *BMVC*, 2018. 4

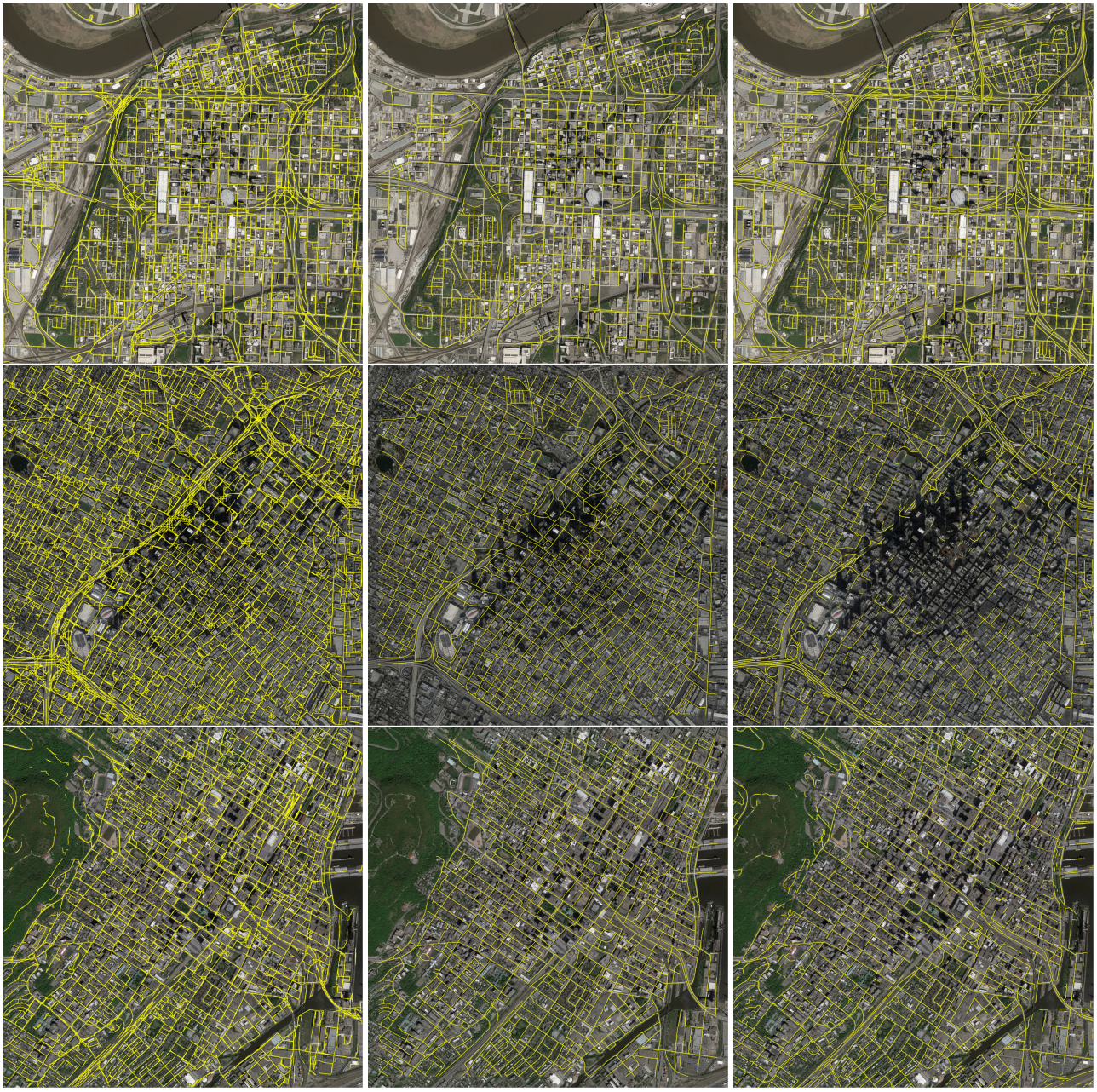

Figure 3. More qualitative results of our method (left) compared to RoadTracer [3] (middle) and DeepRoadMapper [11] (right), on the RoadTracer test set.

Figure 4. More qualitative results of our method (left) compared to RoadTracer [3] (middle) and DeepRoadMapper [11] (right), on the RoadTracer test set.