Supplemental Material : Probabilistic Image-Driven Traffic Modeling via Remote Sensing

Scott Workman and Armin Hadzic

DZYNE Technologies

This document contains additional details and experiments related to our methods.

1 Dynamic Traffic Speeds++ (DTS++)

We introduce DTS++, an extension of the Dynamic Traffic Speeds (DTS) dataset [5] to include an additional city, Cincinnati, OH. To construct DTS++, we emulated the process used to create DTS, which we outline here. First, we obtained a year of historical traffic speed data (2018) from Uber Movement Speeds [1], a dataset of publicly available aggregated speed data derived from Uber rideshare trips. The speed data is provided as averages over road segments at an hourly resolution, with the underlying road network described using OpenStreetMap. Figure 1 gives a visual overview of Cincinnati's spatial coverage (traffic speeds averaged across time).

Fig. 1: Example traffic speed data from DTS++ for Cincinnati (road segment speeds averaged over time). Green (red) corresponds to fast (slow) traffic speeds.

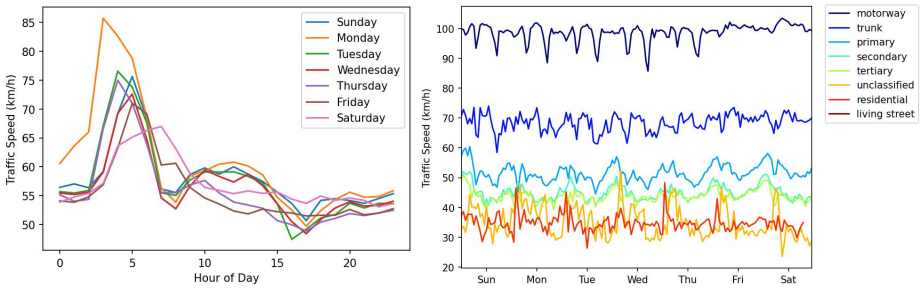

Fig. 2: (left) Visualizing the average road segment speed over time in Cincinnati for different days. (right) Visualizing the average road segment speed against time in Cincinnati, where each road is categorized by its OpenStreetMap road type classification.

To align the historical traffic speed data with overhead imagery, we first generated a set of non-overlapping spherical Mercator tiles (XYZ style) that intersect a bounding box around Cincinnati. We identified tiles containing road segments with available speed data, and for each, downloaded an overhead image from Bing Maps (at a resolution of approximately 0.3 meters per-pixel). This process resulted in 11,137 overhead images (1024×1024) with associated road geometries and traffic speed data. To support model training and evaluation, these non-overlapping tiles are randomly split into 85% training, 5% validation, and 10% testing, such that road segments observed during testing are not observed during training.

In Figure 2 (left), we visualize the average road segment speed versus time for individual days of the week. Additionally, Figure 2 (right) shows the average road segment speed for different types of roads using OpenStreetMap's road type classification. DTS++ extends the DTS dataset by adding a new city, Cincinnati, essentially doubling the size of the dataset and providing support for location adaptation experiments. Compared to New York City (from DTS), Cincinnati has very different appearance characteristics as well as spatiotemporal mobility patterns. For example, the maximum observed traffic speed in Cincinnati for a motorway is ~ 100 km/h, where in New York City it is ~ 80 km/h.

2 Fine-Grained Visual Categorization

We investigate the performance of our geo-temporal positional encoding (GTPE) module on a different task, fine-grained visual categorization, using the iNaturalist 2018 dataset [4]. This dataset contains over 450,000 images of birds (approximately 8000 unique species) collected from the iNaturalist social network. The task is to perform large-scale, fine-grained species classification in the scenario where there are many visually similar species and high class imbalance.

For this experiment, we take advantage of the framework introduced by Aodha et al. [3]. The authors proposed to learn a spatiotemporal prior, $P(y|\boldsymbol{x})$, conditioned on location and time context, $\boldsymbol{x}$, that estimates the probability that

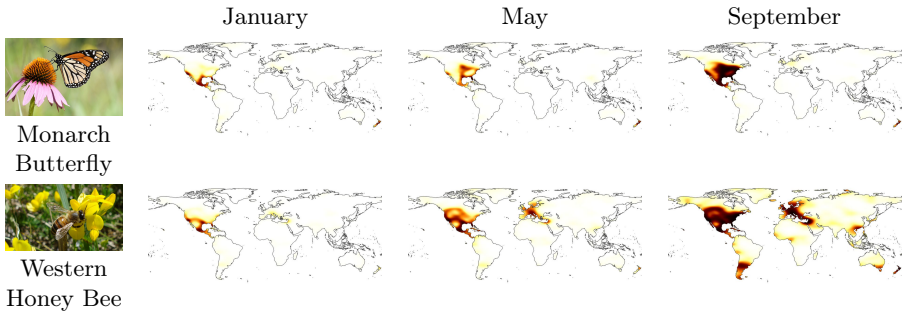

Fig. 3: Visualizing the spatiotemporal prior, learned using the GTPE module, for several species in the iNaturalist 2018 dataset. Darker colors correspond to higher likelihood of the species appearing at that location.

a given object category, y, occurs at that location. The spatiotemporal prior is combined with the output of an image classifier, $P(y|I)$. The results show that including the spatiotemporal prior helps disambiguate fine-grained categories. We take advantage of this framework, but use our proposed geo-temporal positional encoding module to learn the spatiotemporal prior, instead of their approach which uses a MLP consisting of several residual layers and a final output embedding layer (each of 256 dimensions).

We start from public code made available by the authors. The only changes we make to the base training configuration are to set the learning rate to 10^{-4} and remove the learning rate decay policy (for fairness across methods). When integrating GTPE, we use the same parameterization for location/time as the authors and set the hidden/output dims to 256 to follow precedent. Note that our goal here is not to achieve state-of-the-art performance on this benchmark, but rather to understand the impact of our GTPE module against a seminal approach for encoding location/time context.

Table 1 shows the result of this experiment in terms of Top-1, Top-3 and Top-5 accuracy. For all methods, we keep training parameters consistent, only changing which context (location, *loc*, or time, *time*) is made available. For GTPE, *loc*, *time*, and *loc+time*, correspond to the enabling or disabling of individual pathways inside the module. GTPE outperforms the baseline, including both a variant trained locally with the same training configuration, and the original results presented in the paper. Figure 3 shows an example of the spatiotemporal distributions learned by GTPE for several species.

3 Additional Experiments

3.1 Extended Ablation

To evaluate our choice of backbone architecture, we trained a variant of our approach replacing our encoder (which combines convolutional layers and attention

Table 1: Ablation study of our geo-temporal positional encoding on iNaturalist 2018.

Method	Context	Top-1	Top-3	Top-5
[3] (paper)	*loc*	72.41	87.19	90.60
	loc, time	72.68	87.26	90.79
[3] (local)	*loc*	71.62	86.62	90.27
	loc, time	71.53	86.75	90.28
Ours	*loc*	72.64	87.21	**90.80**
	loc, time	72.66	87.28	90.70
	loc, time, loc+time	**72.73**	**87.31**	90.75

layers) with the hierarchical transformer encoder from SegFormer [6], omitting contextual inputs (location and time). This is equivalent to our image-only variant. When evaluated on New York City, the performance of this baseline is 10.51 RMSE, 7.79 MAE, and 0.48 R^2, which is worse than our image-only variant (10.28, 7.70, and 0.50 respectively), but inline with expectations. Though these results show our choice of backbone architecture to be superior for image-driven traffic modeling, we would note that our proposed components (e.g., GTPE, probabilistic formulation) could conceivably be combined with any backbone architecture.

3.2 Visualizing Geo-Temporal Context

Our method integrates location and time through distinct pathways in the GTPE module which can be thought of as 64-dimensional embeddings. To visualize the learned feature representations for time, we construct a false color image using principal component analysis (PCA), shown in Figure 4. For every time combination (day of week, hour of day), we extract its feature representation from the time embedding, and stack them. This results in a matrix of size 168×64. We apply PCA to reduce the feature dimensionality and reshape to 7×24 (days by hours). The resulting image represents time-dependent traffic patterns learned by our approach.

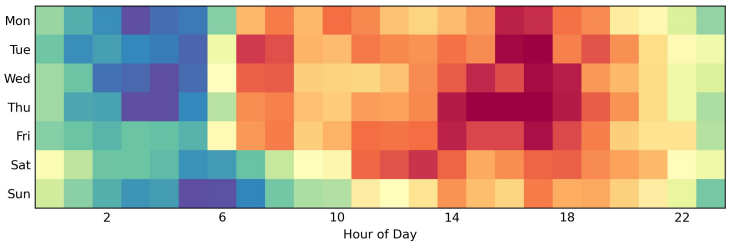

Fig. 4: Visualizing the time embedding learned by GTPE.

As observed, red correlates with higher traffic activity and blue depicts lower. For example, there is less activity in the evenings (midnight-7am), with an increase in activity at 8am across all days as people tend to become active at that time. The visualization also shows that the time embedding captures additional trends, such as: 1) peak traffic around 5pm-6pm across all days, 2) Sunday having reduced traffic throughout the entire day, and 3) increased activity early on Saturday and Sunday mornings. Overall, this visualization shows that our proposed GTPE module is learning to relate geo-temporal context to mobility patterns.

3.3 Capturing Uncertainty in Traffic Speeds

Our method implicitly models uncertainty in traffic speeds at a location/time. This is due to our probabilistic formulation, where instead of regressing traffic speeds directly, we estimate prior distributions over traffic speeds. The output of our approach are the location and scale parameters of a (per-pixel) Student's t-distribution, which are aggregated across an individual road segment and combined with the observed count of traffic observations to form a per-road-segment Student's t-distribution. During model training, we treat the ground-truth traffic speed for a given segment as a sample from the estimated distribution, and minimize negative log-likelihood. Figure 5 visualizes how our approach captures the underlying uncertainty in traffic speed for a given road segment.

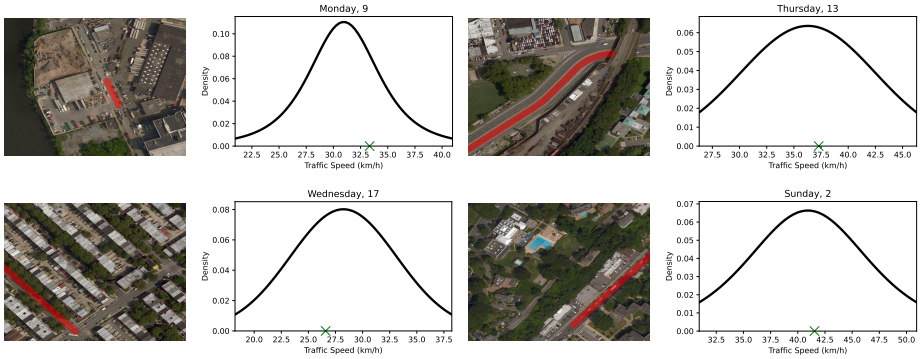

Fig. 5: Our method outputs prior distributions over traffic speeds that implicitly capture uncertainty. (right) Output from our approach visualized as a probability density function corresponds to the (left) road segment depicted in the image. The green x represents the ground-truth traffic speed at the given time.

3.4 Auxiliary Tasks

Figure 6 shows example outputs for road segmentation and orientation estimation. Given an overhead image, our method is able to identify roads and di-

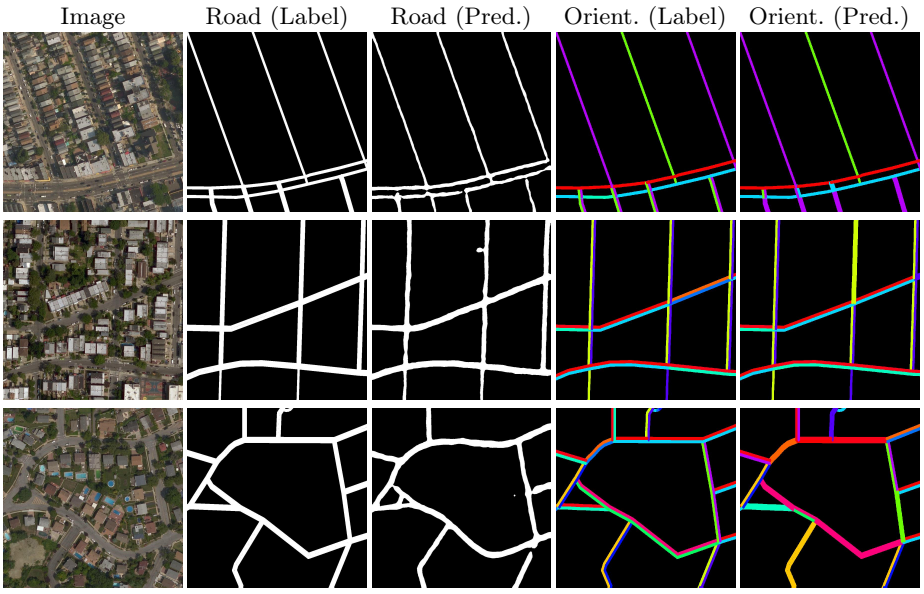

Fig. 6: Qualitative results for the auxiliary tasks of road segmentation and orientation estimation. Orientation is represented by $K = 16$ bins using the HSV color map.

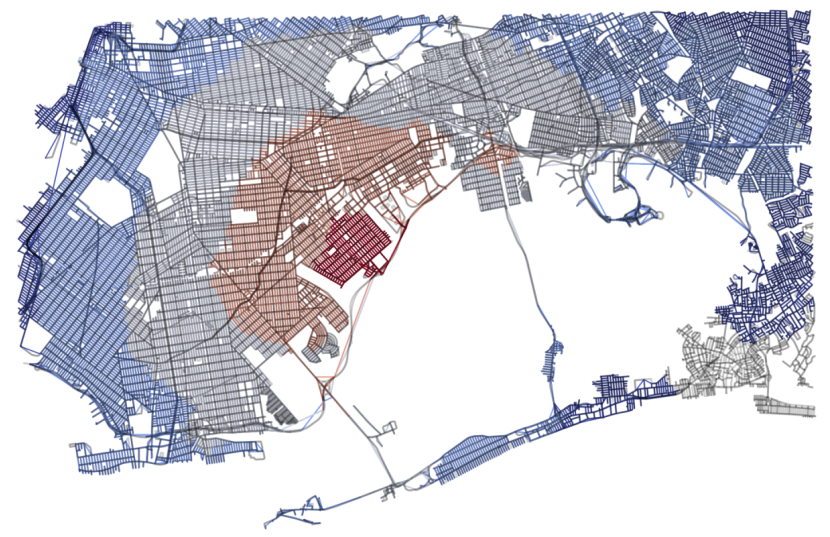

Fig. 7: The output of our approach can be used for various applications, such as generating isochrone maps reflecting travel time. This map reflects travel times on a Monday at 8am. Starting from the center, each band reflects areas reachable in five minute increments (dark red shortest, dark blue furthest).

rections of travel. Combining these outputs with our estimates of traffic speeds enables our approach to estimate local motion models that describe traffic patterns. Finally, Table 2 shows quantitative results for the auxiliary tasks. Though our objective was not necessarily to produce the best road detector or orientation estimator, these results show that our method is capable of capturing a notion of traversability.

Table 2: Evaluation of auxiliary tasks.

Speed (RMSE ↓)	Road (F1 Score ↑)	Orientation (Accuracy ↑)
8.84	77.51%	73.78%

3.5 Complexity Analysis

Our full method has approximately 18.1 million trainable parameters and the total estimated model parameters size is 72.57 MB. It takes about 48 hours to train on a single NVIDIA A6000 (for fair comparison with baselines, we use a batch size of two and accumulate gradients across eight batches). We estimate that our approach has 1587.21 GFLOPS (floating point operations) and 792.41 GMACs (multiply-add operations). During inference, our unoptimized implementation achieves approximately 5.2 frames per second (batch size of 1).

4 Application: Estimating Travel Time

Our approach can be used to generate dense city-scale traffic models (in space and time). One potential application of this is generating travel time maps. We demonstrate this using OSMnx [2], a library for visualizing street networks from OpenStreetMap. We correlate the output from our approach with the graph topology for New York City from OSMnx, using road segment lengths and our estimated speeds to assign travel times to each edge. Figure 7 shows one such result, visualized as an isochrone map highlighting regions with similar travel time.

5 Detailed Architecture

We provide detailed architecture descriptions for the components of our network. Table 3 shows the feature encoder. Table 4 shows our MLP decoder used for generating the segmentation output. Finally, Table 5 shows the geo-temporal positional encoding module.

Table 3: Encoder architecture.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Encoder	[1, 3, 1024, 1024]	–	[1, 64, 256, 256]	–
— Sequential: 1-1	[1, 3, 1024, 1024]	–	[1, 64, 512, 512]	–
— — Conv2d: 2-1	[1, 3, 1024, 1024]	[3, 3]	[1, 48, 512, 512]	1,296
— — BatchNorm2d: 2-2	[1, 48, 512, 512]	–	[1, 48, 512, 512]	96
— — ReLU: 2-3	[1, 48, 512, 512]	–	[1, 48, 512, 512]	–
— — Conv2d: 2-4	[1, 48, 512, 512]	[3, 3]	[1, 64, 512, 512]	27,648
— — BatchNorm2d: 2-5	[1, 64, 512, 512]	–	[1, 64, 512, 512]	128
— — ReLU: 2-6	[1, 64, 512, 512]	–	[1, 64, 512, 512]	–
— — Conv2d: 2-7	[1, 64, 512, 512]	[3, 3]	[1, 64, 512, 512]	36,864
— BatchNorm2d: 1-2	[1, 64, 512, 512]	–	[1, 64, 512, 512]	128
— ReLU: 1-3	[1, 64, 512, 512]	–	[1, 64, 512, 512]	–
— MaxPool2d: 1-4	[1, 64, 512, 512]	3	[1, 64, 256, 256]	–
— ModuleList: 1-5	–	–	–	–
— — MBConvBlock: 2-8	[1, 64, 256, 256]	–	[1, 64, 256, 256]	44,688
— — MBConvBlock: 2-9	[1, 64, 256, 256]	–	[1, 64, 256, 256]	44,688
— ModuleList: 1-6	–	–	–	–
— — MBConvBlock: 2-10	[1, 64, 256, 256]	–	[1, 128, 128, 128]	61,200
— — MBConvBlock: 2-11	[1, 128, 128, 128]	–	[1, 128, 128, 128]	171,296
— — MBConvBlock: 2-12	[1, 128, 128, 128]	–	[1, 128, 128, 128]	171,296
— ContextEncoder: 1-7	[1, 2, 1024, 1024]	–	[1, 64, 1024, 1024]	–
— — LocationEncoder: 2-13	[1, 2, 1024, 1024]	–	[1, 64, 1024, 1024]	12,736
— — TimeEncoder: 2-14	[1, 4]	–	[1, 64]	12,800
— — LocationTimeEncoder: 2-15	[1, 2, 1024, 1024]	–	[1, 64, 1024, 1024]	42,304
— Conv2d: 1-8	[1, 64, 1024, 1024]	[3, 3]	[1, 256, 1024, 1024]	147,712
— ModuleList: 1-9	–	–	–	–
— — MHSABlock: 2-16	[1, 128, 128, 128]	–	[1, 4096, 256]	1,213,704
— — MHSABlock: 2-17	[1, 4096, 256]	–	[1, 4096, 256]	918,024
— — MHSABlock: 2-18	[1, 4096, 256]	–	[1, 4096, 256]	918,024
— — MHSABlock: 2-19	[1, 4096, 256]	–	[1, 4096, 256]	918,024
— — MHSABlock: 2-20	[1, 4096, 256]	–	[1, 4096, 256]	918,024
— Conv2d: 1-10	[1, 64, 1024, 1024]	[3, 3]	[1, 512, 1024, 1024]	295,424
— ModuleList: 1-11	–	–	–	–
— — MHSABlock: 2-21	[1, 256, 64, 64]	–	[1, 1024, 512]	4,363,784
— — MHSABlock: 2-22	[1, 1024, 512]	–	[1, 1024, 512]	3,182,600

Table 4: Decoder architecture.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Decoder	[1, 64, 256, 256]	–	[1, 1, 1024, 1024]	–
— MLP: 1-1	[1, 512, 32, 32]	–	[1, 1024, 512]	–
— — Linear: 2-1	[1, 1024, 512]	–	[1, 1024, 512]	262,656
— MLP: 1-2	[1, 256, 64, 64]	–	[1, 4096, 512]	–
— — Linear: 2-2	[1, 4096, 256]	–	[1, 4096, 512]	131,584
— MLP: 1-3	[1, 128, 128, 128]	–	[1, 16384, 512]	–
— — Linear: 2-3	[1, 16384, 128]	–	[1, 16384, 512]	66,048
— MLP: 1-4	[1, 64, 256, 256]	–	[1, 65536, 512]	–
— — Linear: 2-4	[1, 65536, 64]	–	[1, 65536, 512]	33,280
— Sequential: 1-5	[1, 2048, 256, 256]	–	[1, 512, 256, 256]	–
— — Conv2d: 2-5	[1, 2048, 256, 256]	[1, 1]	[1, 512, 256, 256]	1,048,576
— — BatchNorm2d: 2-6	[1, 512, 256, 256]	–	[1, 512, 256, 256]	1,024
— — ReLU: 2-7	[1, 512, 256, 256]	–	[1, 512, 256, 256]	–
— Dropout2d: 1-6	[1, 512, 256, 256]	–	[1, 512, 256, 256]	–
— Conv2d: 1-7	[1, 512, 256, 256]	[1, 1]	[1, 1, 256, 256]	513

Table 5: Geo-temporal positional encoding (GTPE) module.

Layer (type:depth-idx)	Input Shape	Output Shape	Param #
GTPE	[1, 2, 1024, 1024]	[1, 64, 1024, 1024]	–
— LocationEncoder: 1-1	[1, 2, 1024, 1024]	[1, 64, 1024, 1024]	–
— — LocParam: 2-1	[1, 2, 1024, 1024]	[1, 3, 1024, 1024]	–
— — SirenNet: 2-2	[1048576, 3]	[1048576, 64]	–
— — — ModuleList: 3-1	–	–	–
— — — — Siren: 4-1	[1048576, 3]	[1048576, 64]	256
— — — — — Linear: 5-1	[1048576, 3]	[1048576, 64]	256
— — — — — Sine: 5-2	[1048576, 64]	[1048576, 64]	–
— — — — Siren: 4-2	[1048576, 64]	[1048576, 64]	4,160
— — — — — Linear: 5-3	[1048576, 64]	[1048576, 64]	4,160
— — — — — Sine: 5-4	[1048576, 64]	[1048576, 64]	–
— — — — Siren: 4-3	[1048576, 64]	[1048576, 64]	4,160
— — — — — Linear: 5-5	[1048576, 64]	[1048576, 64]	4,160
— — — — — Sine: 5-6	[1048576, 64]	[1048576, 64]	–
— — — Siren: 3-2	[1048576, 64]	[1048576, 64]	4,160
— — — — Linear: 4-4	[1048576, 64]	[1048576, 64]	4,160
— — — — Identity: 4-5	[1048576, 64]	[1048576, 64]	–
— TimeEncoder: 1-2	[1, 4]	[1, 64]	–
— — TimeParam: 2-3	[1, 4]	[1, 4]	–
— — SirenNet: 2-4	[1, 4]	[1, 64]	–
— — — ModuleList: 3-3	–	–	–
— — — — Siren: 4-6	[1, 4]	[1, 64]	320
— — — — — Linear: 5-7	[1, 4]	[1, 64]	320
— — — — — Sine: 5-8	[1, 64]	[1, 64]	–
— — — — Siren: 4-7	[1, 64]	[1, 64]	4,160
— — — — — Linear: 5-9	[1, 64]	[1, 64]	4,160
— — — — — Sine: 5-10	[1, 64]	[1, 64]	–
— — — — Siren: 4-8	[1, 64]	[1, 64]	4,160
— — — — — Linear: 5-11	[1, 64]	[1, 64]	4,160
— — — — — Sine: 5-12	[1, 64]	[1, 64]	–
— — — Siren: 3-4	[1, 64]	[1, 64]	4,160
— — — — Linear: 4-9	[1, 64]	[1, 64]	4,160
— — — — Identity: 4-10	[1, 64]	[1, 64]	–
— LocationTimeEncoder: 1-3	[1, 2, 1024, 1024]	[1, 64, 1024, 1024]	–
— — LocParam: 2-5	[1, 2, 1024, 1024]	[1, 3, 1024, 1024]	–
— — TimeParam: 2-6	[1, 4]	[1, 4]	–
— — SirenNet: 2-7	[1048576, 7]	[1048576, 64]	–
— — — ModuleList: 3-5	–	–	–
— — — — Siren: 4-11	[1048576, 7]	[1048576, 128]	1,024
— — — — — Linear: 5-13	[1048576, 7]	[1048576, 128]	1,024
— — — — — Sine: 5-14	[1048576, 128]	[1048576, 128]	–
— — — — Siren: 4-12	[1048576, 128]	[1048576, 128]	16,512
— — — — — Linear: 5-15	[1048576, 128]	[1048576, 128]	16,512
— — — — — Sine: 5-16	[1048576, 128]	[1048576, 128]	–
— — — — Siren: 4-13	[1048576, 128]	[1048576, 128]	16,512
— — — — — Linear: 5-17	[1048576, 128]	[1048576, 128]	16,512
— — — — — Sine: 5-18	[1048576, 128]	[1048576, 128]	–
— — — Siren: 3-6	[1048576, 128]	[1048576, 64]	8,256
— — — — Linear: 4-14	[1048576, 128]	[1048576, 64]	8,256
— — — — Identity: 4-15	[1048576, 64]	[1048576, 64]	–

References

1. Uber Movement: Speeds calculation methodology. Tech. rep., Uber Technologies, Inc. (2019)
2. Boeing, G.: OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. Computers, Environment and Urban Systems **65**, 126–139 (2017)
3. Mac Aodha, O., Cole, E., Perona, P.: Presence-only geographical priors for fine-grained image classification. In: IEEE International Conference on Computer Vision (2019)
4. Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., Belongie, S.: The iNaturalist species classification and detection dataset. In: IEEE Conference on Computer Vision and Pattern Recognition (2018)
5. Workman, S., Jacobs, N.: Dynamic traffic modeling from overhead imagery. In: IEEE Conference on Computer Vision and Pattern Recognition (2020)
6. Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J.M., Luo, P.: SegFormer: Simple and efficient design for semantic segmentation with transformers. In: Advances in Neural Information Processing Systems (2021)