This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Learning to Localize Through Compressed Binary Maps

Xinkai Wei[1,2*] Ioan Andrei Bârsan[1,3*] Shenlong Wang[1,3*]
Julieta Martinez[1] Raquel Urtasun[1,3]
[1]Uber Advanced Technologies Group [2]University of Waterloo [3]University of Toronto
{xinkai.wei,andreib,slwang,julieta,urtasun}@uber.com

Abstract

One of the main difficulties of scaling current localization systems to large environments is the on-board storage required for the maps. In this paper we propose to learn to compress the map representation such that it is optimal for the localization task. As a consequence, higher compression rates can be achieved without loss of localization accuracy when compared to standard coding schemes that optimize for reconstruction, thus ignoring the end task. Our experiments show that it is possible to learn a task-specific compression which reduces storage requirements by two orders of magnitude over general-purpose codecs such as WebP without sacrificing performance.

1. Introduction

One of the fundamental tasks in autonomous driving is the ability to localize the self-driving vehicle (SDV) with respect to a geo-referenced map, as this enables routing the vehicle from point A to point B. Furthermore, high precision localization enables the use of high definition (HD) maps that capture the static parts of the environment. This map is used by most self driving teams as a component of perception and motion planning modules.

LiDAR-based localization systems are usually employed for precise localization of SDVs [3, 9, 12, 13, 30, 31]. They rely on having an HD map, which contains dense point clouds [29, 31] and/or intensity LiDAR images of the ground [3, 12, 13, 30]. One of the main difficulties of scaling current localization systems to large environments is the on-board storage required for the HD maps. For instance, storing a LiDAR intensity map as a 16-bit PNG file would require roughly 900 GB for a city such as Los Angeles, and over 168 TB for the entire United States.[1] Storing this information onboard the vehicle is infeasible for scalability past a single city. Streaming the HD map data on the go makes the system dependent on a reliable broadband connection, which may not always be available.

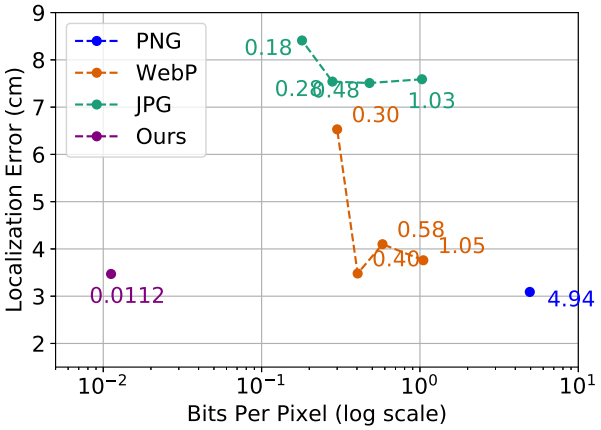

Figure 1: **End failure rate for localization under different map compression settings.** Lower is better.

In this paper we propose to learn to compress the map representation such that it is optimal for the localization task. As a consequence, higher compression rates can be achieved without loss of localization accuracy and robustness compared to standard coding schemes that optimize for reconstruction, thus ignoring the end task. In particular, we leverage a fully convolutional network to learn to binarize the map features, and further compress the binarized representation using run-length encoding on top of Huffman coding. Both the binarization net and the decoder are learned end-to-end using a task-specific loss. We demonstrate the effectiveness of this idea in the context of a state-of-the-art LiDAR intensity-based localization system [3], and show that it is possible to learn a task-specific compression scheme which reduces storage requirements by two orders of magnitude over general-purpose codecs such as WebP, without sacrificing performance.

2. Related Work

Localization Using HD Maps: High-definition maps have been widely used in the field of robot localization due

*Equal contribution

[1]Based on information from the US Bureau of Transportation Statistics, assuming that it takes approximately 4 MB to store a 150 m road segment as a 16-bit PNG single-channel image (https://www.bts.gov).

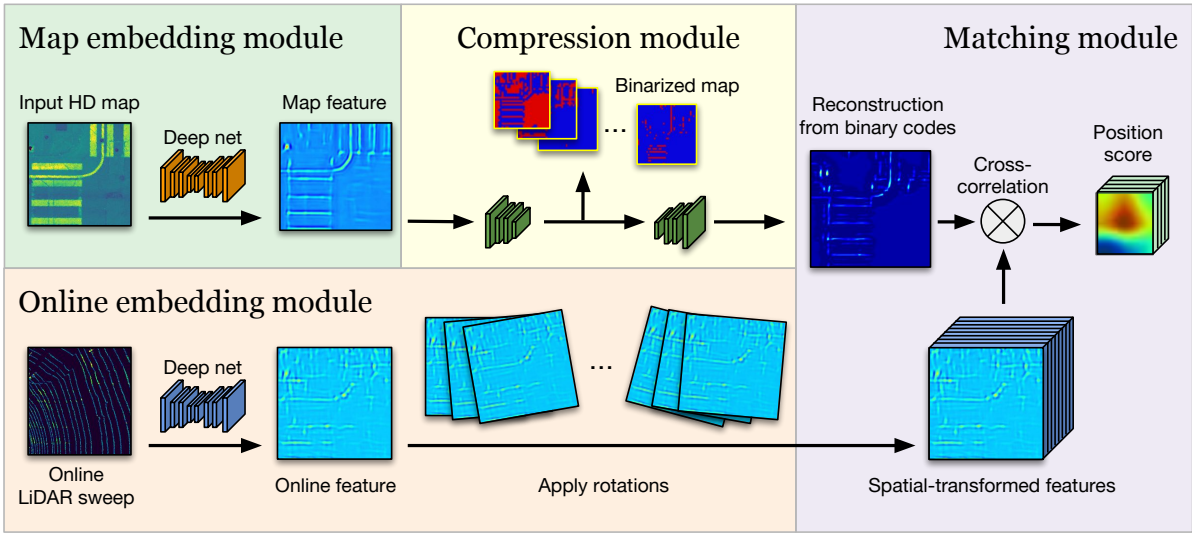

Figure 2: **Architecture overview of our proposed joint compression and localization network.**

to their ability to enable centimeter-level accuracy in a diverse set of environments, while avoiding some of the computational costs typically associated with a full SLAM system. Levinson and Thrun [12] used Graph-SLAM to aggregate LiDAR observations into a coherent map, which was then used in localization. Kümmerle et al. [11] use Multi-Level Surface Maps [26] with LiDAR, and use the map for localization and path planning to enable a car to park itself. Subsequent works have improved the robustness of such approaches by augmenting the maps with probabilistic occupancy information [13, 30], or by fusing LiDAR matching results with differential GPS [27].

Lightweight Localization: Numerous alternatives to HD maps have been explored over the years in an attempt to overcome their limitations, such as the dependence on data collection and offline map construction. Floros et al. [6] develop a lightweight extension to visual odometry which leverages OpenStreetMap to eliminate the drift typically associated with pure VO. Ma et al. [15] extend this idea by using cues from multiple modalities, such as egocar trajectory, road type and the position of the sun to localize robustly within a lightweight map represented as a graph. Recently, Ort et al. [17] used a similar approach, performing road segmentation using LiDAR to localize within a lightweight topological map with negligible on-board map storage requirements. Javanmardi et al. [9] extract probabilistic 3D planar surfaces and 2D vector maps from dense point clouds to create lightweight maps which are then used in localization, achieving promising results in terms of accuracy. In recent years, image-based localization methods [5, 18, 22] have shown promising results reaching centimeter-level accuracy in indoor environments. However, these methods are still not sufficient for centimeter-level accuracy in outdoor scenarios exhibiting fast motion, such as those occurring in self-driving.

Image Compression: Image compression is a classical subfield of computer vision and signal processing. It has seen a great deal of progress in the past few years thanks to the advent of deep learning. In most modern incarnations, a learning-based compression method consists of an encoder network, a quantization mechanism (e.g., binary codes), and a decoder network which reconstructs the input from the quantized codes. Recent learning-based approaches [16, 20, 25] consistently outperform classic compression methods like JPEG2000 and BPG. While earlier works on learned compression typically used a standard autoencoder architecture [2], thereby imposing a fixed code size for all images, Toderici et al. [24] overcome this limitation by using a recurrent neural network as an encoder. Subsequently, Toderici et al. [25] extend the previous results, which were typically presented on resolutions of 64×64 or less due to performance considerations, showing state-of-the-art compression rates on full-resolution images. Recently, Mentzer et al. [16] proposed a pipeline which obtained results on par with the state-of-the-art, while also being trainable end-to-end (encoder, quantizer, decoder).

Learning to Match: Learning-based approaches have also been employed in matching problems, which arise in numerous vision applications, including stereo matching [14, 32], optical flow [19, 28], and map-based localization [3]. In their pioneering work, Zbontar and LeCun [32] proposed modeling the matching cost function used in stereo depth estimation as a convolutional neural network and learning it from data. Luo et al. [14] extend this framework to produce calibrated probability distributions over the disparities of all pixels, while also enabling real-time operation through the introduction of an

explicit correlation layer capable of speeding up inference by an order of magnitude compared to previous work. Similarly, DeepFlow [28] applies learning-based matching to the problem of optical flow estimations. The authors use a learned matcher to match sparse descriptors, which are then fed into a variational method to estimate dense flow. EpicFlow [19] extends this framework by densifying the sparse matches using a novel interpolation scheme before performing variational energy minimization. The task of map-based localization using matching has also been approached from a data-driven perspective, using neural networks to learn representations optimal for matching a LiDAR observation to a map [3].

3. End-to-End Compressed Localization

LiDAR based localization systems are usually employed by self-driving vehicles to provide high-precision localization estimates [3, 13, 30]. They rely on having an HD map, which contains dense point clouds [31] and/or LiDAR intensity images [13, 30]. One of the main difficulties of scaling localization to large environments is the on-board storage required for these maps. To tackle this problem, in this paper we propose to learn to compress the map representation such that it is optimal for the localization task. As a consequence, higher compression rates can be achieved without loss of localization accuracy or robustness degradation. In particular, our approach learns a compressed deep embedding of the map that can be directly stored on-board, dramatically reducing the requirements of state-of-the-art LiDAR intensity based localization systems.

In this section, we first revisit the state-of-the-art Deep Ground Intensity Lidar Localizer (Deep GILL) [3] and its probabilistic Bayes inference. We then describe our compression module and show how it can be learned end-to-end jointly with the localizer.

3.1. Deep GILL Revisit

Real-time localization with centimeter level accuracy is critical for most self-driving cars, as they rely on the semantics captured in HD maps to drive safely. In this paper, we follow [3]'s formulation for our localization as a recursive Bayes inference problem. In particular, the Bayes inference framework combines the LiDAR matching energy, the vehicle dynamics, and the GPS observations with the estimates of the previous time step to form the probability of a given location at the current time step. Let $\mathrm{Bel}(\mathbf{x}_t)$ be the posterior distribution of the vehicle pose at time t given all the sensor observations until time step t, we have:

$$\mathrm{Bel}_t(\mathbf{x}) = \mathrm{Bel}_{t|t-1}(\mathbf{x}; \mathcal{X}) \cdot P_{\text{GPS}}(\mathcal{G}_t|\mathbf{x}) \cdot P_{\text{LiDAR}}(\mathcal{I}_t|\mathbf{x}; \mathbf{w}), \tag{1}$$

where $\mathbf{x} = \{t_x, t_y, \theta\}$ is the 3-DoF vehicle pose, and $\mathcal{I}_t \in \mathbb{R}^{N_t \times 4}$ is the online LiDAR sweep containing N_t points with geometric and intensity information; $\mathcal{X}_t = \mathbf{v}_\mathbf{x}, \mathbf{v}_\theta$ is the vehicle dynamics encoding linear and angular velocity; and $\mathcal{G}_t \in \mathbb{R}^2$ are GPS observations under Universal Transverse Mercator (UTM) coordinate system.

The motion model encodes the fact that the inferred pose should agree with the vehicle dynamics given the previous time step location belief $\mathrm{Bel}_{t-1}(\mathbf{x}_{t-1})$, more formally defined as

$$\mathrm{Bel}_{t|t-1}(\mathbf{x}|\mathcal{X}_t) = \int_{\mathbf{x}_{t-1}} P(\mathbf{x}|\mathcal{X}_t, \mathbf{x}_{t-1}) \mathrm{Bel}_{t-1}(\mathbf{x}_{t-1}). \tag{2}$$

We use a Gaussian to represent the vehicle's conditional distribution over vehicle dynamics,

$$P(\mathbf{x}|\mathcal{X}_t, \mathbf{x}_{t-1}) \propto \mathcal{N}(\mathbf{x}_{t-1} \oplus \mathcal{X}_t, \Sigma), \tag{3}$$

where $\mathbf{x}_{t-1} \oplus \mathcal{X}_t$ is the last timestamp's pose composed by the current timestamp's velocity observation; $\oplus$ is the pose composition operator, and Σ is the covariance matrix for the velocity estimation. The GPS observation model encodes the likelihood of the GPS as a Gaussian distribution:

$$P_{\text{GPS}} \propto \mathcal{N}([g_x, g_y]^T, \sigma^2_{\text{GPS}}\mathbf{I}), \tag{4}$$

where g_x and g_y is the map-relative position $\mathbf{x}$ converted to UTM coordinate system.

The LiDAR matching model encodes the agreement between the current online LiDAR observation and the map indexed at the hypothesized pose $\mathbf{x}$:

$$P_{\text{LiDAR}} \propto s\left(\pi\left(f(\mathcal{I}; \mathbf{w}_{\text{O}}), \mathbf{x}\right), g(\mathcal{M}; \mathbf{w}_{\text{M}})\right), \tag{5}$$

where $f(\cdot)$ and $g(\cdot)$ are the embedding network over online LiDAR sweeps and maps respectively, and π is a rigid transform that converts the online embedding image to the map coordinates using the hypothesized pose $\mathbf{x}$; $\mathcal{M}$ is the dense LiDAR intensity map representation, and s is the correlation operator between the online embedding and map embedding. The computation of this term can be written as a feed-forward network as shown in [3].

While effective for localization, the dense intensity map used in Deep GILL [3] requires a large amount of on-board storage. This prevents the method from scaling to larger operational domains. To tackle this problem, in this paper we introduce a novel learning-to-compress module that reduces the storage of the map significantly, allowing us to potentially store maps for the full continent. Next, we describe our new compressed model.

3.2. Deep Localization with Map Compression

Unlike previous compression networks that aim at optimizing the reconstruction error or perceived visual quality, in this paper we argue that optimizing for the tasks at hand

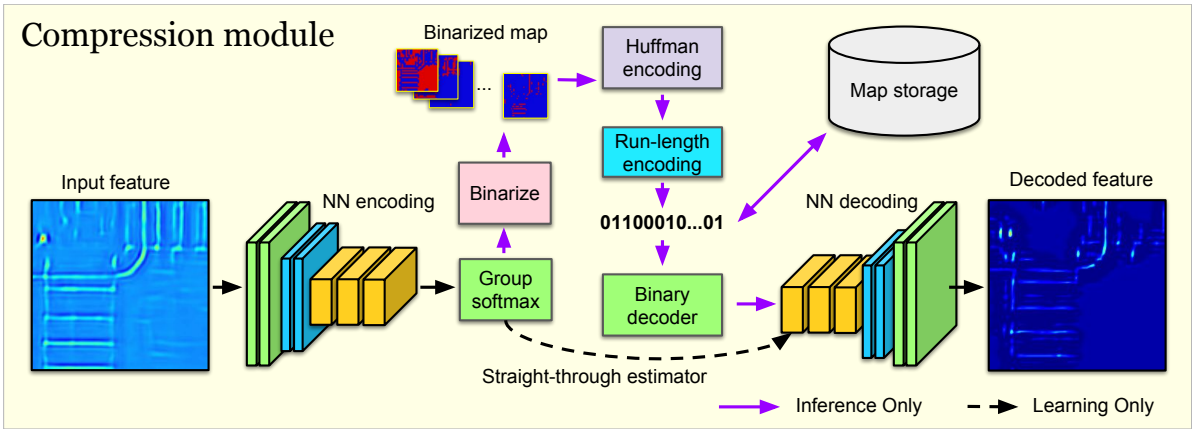

Figure 3: **Our compression module.** We obtain gradients for training with a straight-through estimator.

is important to further reduce the storage requirements. Towards this goal, we extends the architecture of [3] and include a compression module responsible for encoding the map with binary codes through deep convolutional neural networks. Importantly, our encoding can be learned end-to-end with our localizer.

We refer the reader to Fig. 2 for an illustration of the overall architecture of our joint compression and matching network. Our overall end-to-end network includes three components. First, our embedding module takes the map $\mathcal{M}$ and the online LiDAR sweep $\mathcal{I}$ as input, and computes a deep embedding representation of both. A compression module is then applied over the map embedding layer, which converts the high-dimensional float-valued deep embedding map to a compact convolutional binary code representation. This representation is used as a compact storage of the map. A decoding module is then employed to decode the binary codes back to the real-valued embedding representation. Finally, matching is conducted between the reconstructed map embedding and the online embedding. This gives us a score for each possible transformation. We use softmax to build the probability P_{LiDAR} over our localization search space from the raw matching score. We now describe the modules in more detail.

Embedding Module: The embedding should capture robust yet discriminative contextual features while preserving pixel-accurate details for precise matching. Motivated by this fact, we designed this module to be a fully convolutional encoder-decoder network following [3]. It has an U-Net architecture [21]. The encoder consists of four blocks, each of which has two stride-1 3×3 conv layers and one stride-2 3×3 conv layer that down-samples the feature map by a factor of 2. The numbers of channel per each block are 64, 128, 256, 512, respectively. The decoder network has four decoder blocks, each of which takes the last decoder block's output and the corresponding encoder layer feature as input in an additive manner. Each block contains one 3×3 deconv layer followed by one stride-1 3×3 conv. The final embedding map has the same spatial resolution as the input with depth equals to embedding dimension. In this way the decoder combines both high level contextual information as well as low-level details.

Compression Module: We highlight the task-specific map compression module as the core contribution of this paper. The purpose of this module is to convert the large-resolution, high-precision embedding into a low-precision, lower-resolution one, without losing critical information for matching. We employ a fully convolutional encoder-decoder network to achieve this goal. The neural network encoder is a fully convolutional residual network where each scale have two 3×3 standard residual blocks [8] and a stride-2 3×3 conv between scales. The dimensionality per scale is $8, 16, 32, 64$ respectively. The decoder is a fully convolutional network with several transposed convolutional layers. We use the PReLU [7] as the activation function. The output of the encoding module is passed through a grouped soft-max module, with a binarization module defined as

$$p_j = \frac{\exp(f_j)}{\sum_{k \in \mathcal{S}_j} \exp(f_k)}, \quad b_j = \begin{cases} 1 & \text{if } p_j \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\mathcal{S}_j$ is the index group that j belongs to, with each group representing a non-overlapping subset of the full index set $\{1, \ldots, K\}$; $\mathbf{f} = [f_0, \ldots, f_i, \ldots]$ is the input feature. The benefit of using grouped-softmax as encoder activation along with the binarizer is twofold. First, within each group, we have at most one non-zero entry. Thus, with the same number of channels it has better sparsity than the sigmoid function, increasing the compressibility of the binary encoding. Second, compared against standard soft-max, it increases the potential capacity since the grouping of indices allows a more structured encoding. While the component is non-differentiable, backpropagation was still feasible thanks to the use of a straight-through estimator, which we will show in Sec. 3.3 in detail.

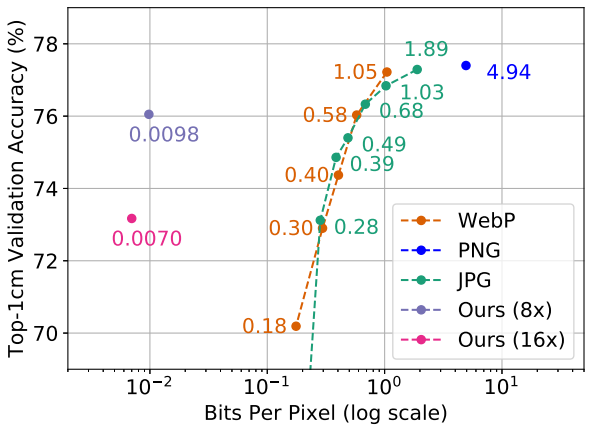

Figure 4: **Top-1 Matching Performance vs Bits Per Pixel**

Method	BPP	Top-9 px	Top-1 px
Lossless (PNG)	4.93	97.47%	**77.40%**
Ours (recon, 8x)	0.0520	97.27%	75.83%
Ours (recon, 16x)	0.0140	96.95%	74.86%
Ours (match, 8x)	0.0098	**97.73%**	76.05%
Ours (match, 16x)	**0.0070**	97.25%	73.17%

Table 1: **Ablation studies on matching performance.** Optimizing jointly for both map reconstruction and matching greatly reduces the memory requirements.

Thus, we only need to store onboard these highly compressed binary map embeddings for localization. A two-step lossless binary encoding scheme is adopted. Our first step is a Huffman encoding. The motivation is that the frequency of appearance of items are not equal. The Huffman dictionary is built by the one-hot encoding of the softmax latent probability $\mathbf{p}$ per pixel. Thus, for a 128-softmax vector the dictionary size is 128. Frequency is computed in a batch manner. For instance, if the 'class' $5(00000101)$ appears 50% we could use a shorter-length code 0 to encode it. After that a run-length encoding (RLE) is conducted over the flatten Huffman code map to further reduce the size by making use of the fact that codes appear consecutively. For instance 5555558 could be further reduced to 5681. This give us the final binary code that we store. Note that both Huffman encoding and run-length encoding are lossless. We choose Huffman+RLE due to its efficiency and effectiveness. Empirically, we also show that this approach reaches 72.5% of the ideal entropy lower bound. While other types of entropy coding, such as arithmetic coding exist, they are slower and bring marginal improvements to compression rates [1].

Combining the NN encoding and the binary encoding, this full compressive encoding scheme gives a very large gain in terms of storage efficiency as shown later in the experimental section.

The decoder module then takes the binary code as input. First, it transforms the Huffman+RLE codes back to the binary map, and then applies a series of deconvolutional blocks to recover the full high resolution, high-precision embedding of the map that we use for matching. Fig. 3 illustrates the pipeline of the full compression module.

Matching Module: Our matching module follows [3], where a series of spatial transformer networks are utilized to rotate the online embedding multiple times at $|\Theta|$ different candidate angles. Within each rotation angle, translational search based on inner-product similarity is equivalent to convolving the map embedding with the online embedding as kernels. Thus, enumerating all the possible pose candidates is equivalent to a convolution with $|\Theta|$ kernels. Unlike standard convolutions, this convolution has a very large kernel. Following [3], we exploit FFT-conv to accelerate this matching modules by an order of magnitude (compared to GEMM-based convolutions) on a GPU.

3.3. End-to-End Learning

Our full localization network is trained end-to-end, as the compression module is active during the training loop. Our loss function consists of two parts, namely a matching loss that encourages that the end-task is accurate and a compression loss that minimizes the encoding length. Thus, our total loss is defined as

$$\ell = \ell_{\text{LOC}}(\mathbf{y}, \mathbf{y}_{\text{GT}}) + \lambda_1 \ell_{\text{MDL}}(\mathbf{p}) + \lambda_2 \ell_{\text{SPARSE}}(\mathbf{p}), \quad (7)$$

where $\mathbf{y}$ is the final softmax-normalized matching score, $\mathbf{y}_{\text{GT}}$ is the one-hot representation of the ground truth (GT) position and $\mathbf{p}$ is the embedding after the grouped-softmax layer in the compression module, defined in Eq. 6.

We employ cross-entropy as a matching loss. This encourages the matching score to be the highest at the GT position, while lowering the score of positions elsewhere:

$$\ell_{\text{LOC}}(\mathbf{y}, \mathbf{y}_{\text{GT}}) = \sum_i y_{\text{GT},i} \log(y_i).$$

The compression loss tries to minimize the encoding length. In particular, we use entropy as a differentiable surrogate of code length. Note that this surrogate has been widely used in previous deep compression approaches [24]. According to Shannon's source coding theorem [23], entropy provides an optimal code length, which could serve as a surrogate lower-bound for the actual encoding that we use. Our entropy is estimated within each mini-batch as

$$\ell_{\text{MDL}}(\mathbf{p}) = \bar{\mathbf{p}} \log \bar{\mathbf{p}},$$

where $\bar{\mathbf{p}} = \frac{1}{W \times H \times B} \sum_i \mathbf{p}_i$ is the mean soft-max probability averaged across all pixels' softmax probability $\mathbf{p}_i$ in one batch example, defined as in Eq. 6. In practice we find that this theoretical lower-bound is very close to the actual bit per pixel rate obtained after Huffman+RLE encoding.

Finally, we want the soft-max probability to be as close to one-hot as possible to reduce the loss due to hard binarization. We thus minimize each individual pixel's entropy as a regularization term:

$$\ell_{\text{SPARSE}}(\mathbf{p}) = \sum_i p_i \log p_i$$

Note that direct backpropagation is not feasible in our case, as the binarization module defined in Eq. 6 is not differentiable. To overcome this issue, we adopt the straight-through estimator proposed in [4]. That is, during the forward pass we conduct hard binarization, while during the backward pass we substitute this module with an identity function. We find that this approximation provides good gradients for the function to be learned.

3.4. Efficient Inference

In the offline map encoding stage, we use our compression network to compress the map into a binary code such that the onboard storage requirements are minimized. During onboard inference, the compressed code is recovered, the decoder is then used to create the HD embedding map, which is used for localization.

Onboard Inference: Computing the exact probability defined in Eq. 1 is not feasible due to the continuous space and the infeasible integral for the vehicle dynamics model defined in Eq. 2. Following [3], we use a histogram filter to approximate the inference process. Towards this goal, we discretize the search space around a local region to a 5×5 cm grid. The integration will only be computed within this local trust-region, which neglects the rest of the solution space where the belief is negligible. In this manner, both the GPS and the dynamics term can be computed efficiently over a local grid as the search space. Unlike [3], when computing the LiDAR matching term $P_{\text{LiDAR}}(\mathcal{I}_t|\mathbf{x};\mathbf{w})$, we first retrieve a local map binary code $\mathbf{b}$. The LiDAR embedding is computed through the feature network and then $\mathbf{b}$ is passed through the decoder of the compressor to recover the map embedding $g(\mathcal{M})$. After that, the matching score P_{LiDAR} can be efficiently computed for all hypothesized poses as a feed-forward network through FFT-conv. After each term has been computed, the final pose estimation is a soft-argmax aggregation taking uncertainty into consideration:

$$\mathbf{x}_t^* = \frac{\sum_{\mathbf{x}} \text{Bel}_t(\mathbf{x})^\alpha \cdot \mathbf{x}}{\sum_{\mathbf{x}} \text{Bel}_t(\mathbf{x})^\alpha} \tag{8}$$

where $\alpha \geq 1$ is a temperature hyper-parameter.

4. Experimental Evaluation

Dataset: We evaluate our approach over two large-scale driving datasets that cover highway and urban driving, respectively. The highway dataset was collected in [3] and contains over 400 sequences of highway driving with a total of 3 000 km travelled. It contains an HD, dense, LiDAR intensity map stored in lossless PNG format. The self-driving vehicle integrates a 64-line LiDAR sweeping at 10Hz, with GPS and IMU sensor. We follow the setting of [3] and select 282 km of driving as testing, ensuring that there is no geographic overlap between the splits. The GT localization is estimated by a high-precision offline Graph-SLAM.

To better evaluate the potential of the model to compress maps with more diverse content and complicated structures, we build a new urban driving dataset that consists of 15 554 km of driving. This dataset is collected in a metropolitan city in North America with diverse scenes and road structures. This dataset is more challenging as it introduces more diverse vehicle maneuvers, including sharp turns and reverse driving, as well as some regions with poor lane markings and map changes. The ground-truth localization is estimated through an high-precision offline Graph-SLAM with multi-sensor fusion. Intensity maps are built by multiple passes through a comprehensive offline pose graph optimization at a resolution of 5cm per pixel.

Experimental Setup: To our knowledge, no previous work has integrated LiDAR intensity localization with deep compression. Therefore, we evaluate our work against baselines without compression on localization metrics alone, such as those found in [3], and measure the performance degradation when using the map compression module.

Since the intensity map is stored as an image, we compare against several traditional image compression algorithms such as JPEG and WebP. For each compression algorithm, we compress the training and testing map images and train a standard learn-to-localize matching network [3]. We also train a reconstruction-based compression network ('ours (recon)') that shares the same architecture with our compression module, with the only exception that it is trained for reconstruction error of the feature map only (not for matching performance). This showcases whether the task-specific compression helps our matching task.

For our proposed method, we adopt two different settings for compresssion, by changing the downsampling levels we used for our binary codes. We have 8x downsampling model and 16x downsampling model, where the 16x model has an extra set of downsampling and upsampling modules before binarization, which varies the compression rate. All the competing algorithms have the same embedding feature network as our proposed model. Additionally, we performed experiments with reduced map resolution as an alternative baseline for reducing map storage.

We train all competing algorithms over 343k and 230k training samples for urban and highway datasets respectively. We aggregate five online LiDAR sweeps and rasterize them into a birds' eye view image at 5cm/pixel, in ranges of (-12 m, 12 m) and (-15 m, 15 m). All networks

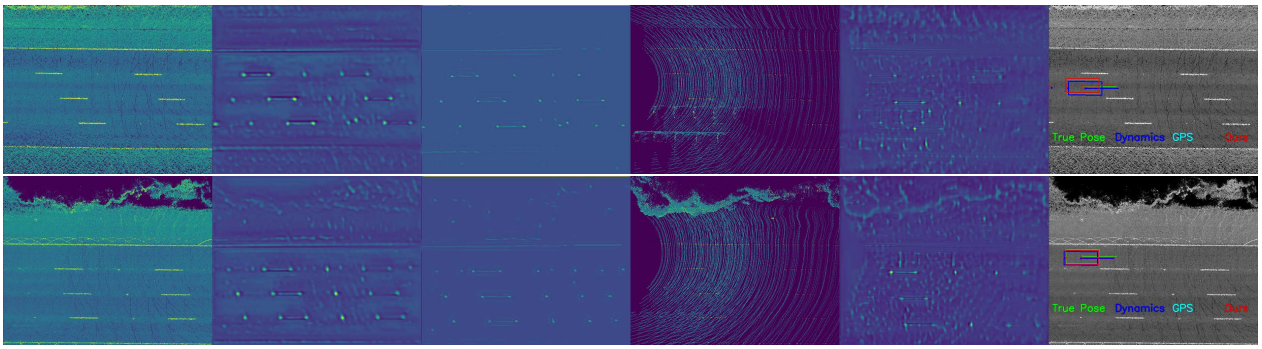

Figure 5: **Qualitative results from our highway dataset.** From left to right: (1) original map, (2) its computed deep embedding, (3) the compressed embedding, (4) online LiDAR observation, (5) its embedding, and (6) the localization result.

Method	**Median error (cm)**			**Failure rate (%)**			**Bit per pixel**
	Lat	Lon	Total	$\leq$ 100m	$\leq$ 500m	End	
Lossless (PNG)	1.55	**2.05**	**3.09**	**0.00**	1.09	2.44	4.94
JPG-5	4.32	5.48	8.41	**0.00**	1.09	1.25	0.18
JPG-10	3.42	5.46	7.54	**0.00**	1.09	5.26	0.28
JPG-20	3.77	4.99	7.51	**0.00**	**0.00**	1.75	0.48
JPG-50	3.29	5.60	7.59	**0.00**	1.09	5.26	1.03
WebP-5	1.65	5.75	6.53	2.04	5.43	13.95	0.30
WebP-10	1.60	2.26	3.48	**0.00**	1.09	2.50	0.40
WebP-20	1.86	2.85	4.10	4.08	8.70	14.63	0.58
WebP-50	1.62	2.75	3.76	**0.00**	3.26	3.30	1.05
Ours	1.61	2.26	3.47	**0.00**	1.09	**1.22**	**0.0083**

Table 2: **Online localization performance on the urban dataset.**

are trained on four NVIDIA 1080 Ti GPUs using PyTorch. We use the Adam optimizer [10] with an initial learning rate of 10^{-3}. We observed that training the entire network end-to-end from scratch works, but is slower to converge. To speed up training, we first train the non-compressed matching network without our additions for the localization task, and then insert our compression module and train end-to-end.

Matching Performance: In order to evaluate the performance of the models in terms of finding the best match in a compressed map, we report the performance of the competing algorithms under the matching setting.

We conduct matching over a 1 m^2 search range, after perturbing the initial position of the vehicle around the GT position. We sample the translational perturbation between 0 and 1 m^2, and the angular perturbation between 0 and 5°, both uniformly. We report top-1 px and top-9 px as our metrics, representing whether the prediction is in the same pixel as the GT or within the 3×3 region centered around the GT, respectively. We report matching accuracy as a function of bit rate per pixel on the urban dataset in Fig. 4. Note that the proposed algorithm at 8x setting achieves 76% top-1 px accuracy with 0.0098 bit per pixel rate. Both are higher than all competing algorithms. Also, it obtains similar top-9 px accuracy on par with no compression module. Especially, the BPP is around 20-400 times smaller than all competing algorithms. Under the 16x setting the top 1 px accuracy is 3% lower but achieves a higher compression rate at 0.007 bits per pixel.

Ablation Studies: We conducted an ablation study over the matching performance. We first validate whether jointly training the compression module with our matching task loss helps improve the matching performance and increase the compression rate. For this, we train a compression module using only reconstruction loss (without the matching task loss). Secondly, we report whether a lower compression rate is achieved through aggressive map downsampling. Table 1 illustrates the results. We can see that jointly training the compression module with the task specific loss greatly helps the performance. The 16x downsampled model pushes the compression rate even further, with a 3% percent drop on top 1px results.

Online Localization: We follow [3] and compute the median and worst case localization error on the test split as our metrics. To be specific, we report median, p95, and p99 error in meters along the lateral and longitudinal directions. We also report an out-of-range rate, which represents the percentage of 1 km segments where the method reaches a localization error of 1m.

Method	Median error (cm)			Failure rate (%)			Bit per pixel
	Lat	Lon	Total	$\leq$ 100m	$\leq$ 500m	End	
Lossless (PNG)	**3.62**	**4.53**	**7.06**	**0.00**	**0.35**	0.72	4.97
WebP-50	3.87	4.87	7.52	**0.00**	0.71	**0.71**	0.58
WebP-20	4.03	5.27	8.02	**0.00**	1.06	8.87	0.36
WebP-10	4.45	7.09	9.79	0.35	9.57	24.37	0.26
WebP-5	4.10	6.40	8.99	0.35	9.57	14.69	0.20
Ours	**3.62**	4.77	7.19	0.35	**0.35**	**0.71**	**0.007**

Table 3: **Online localization performance on the highway dataset.**

Method	Median error (cm)			Failure rate (%)			Bit per pixel
	Lat	Lon	Total	$\leq$ 100m	$\leq$ 500m	End	
Lossless (PNG)	**1.55**	**2.05**	**3.09**	**0.00**	1.09	2.44	4.97
Ours (recon, 8×)	1.59	2.16	3.24	0.00	1.09	1.22	0.027
Ours (recon, 16×)	1.76	2.48	3.62	0.00	0.0	2.56	0.012
Ours (match, 8×)	1.61	2.26	3.47	0.00	1.09	1.22	0.021
Ours (match, 16×)	1.62	2.77	3.84	1.00	2.17	4.26	0.007

Table 4: **Ablation studies on the urban dataset.**

Table 2 shows the online localization performance on the urban dataset. While most of the baselines provide reasonable results, our method is clearly better than competing algorithms such as JPG-5 and WebP-5, which show high failure rates at extreme compression levels. In terms of worst case, measured by failure rate, our method is on par with high-quality compression such as WebP-50 and JPEG-50, and a lossless method, while our bit rate per pixel is 100 times smaller. This is shown in in Fig. 1, where we plot the percentage of failures after 1 km against the storage.

Table 3 depicts the online localization performance on the highway dataset. We can see that traditional off-the-shelf compression algorithms like WebP have a large performance drop compared to our compression-based matching. While the method using reconstruction loss obtains storage roughly in the same magnitude as our approach, it suffers a large performance drop. This indicates the importance of matching loss term for effectively selecting portions of the map to keep. Meanwhile, our method based on the matching task loss has no performance drop at half the storage of the pure reconstruction network, nor at more than 400 times smaller compared to the lossless compression bitrate.

Table 4 showcases the ablation study on the urban dataset. We compare reconstruction loss driven compression models against our matching-loss driven compression model under various architectures. From the Tables we can see that, in terms of online localization error, our compression model trained with task-specific driven loss is better than the reconstruction model with smaller bitrates and a lower failure rate.

Qualitative Analysis: Fig. 5 shows examples of the deep map embeddings computed by our system (before and after compression) together with the (uncompressed) online observation embedding and the localization result. For more results please refer to the supplementary material.

Compression	LA County	Full US
Lossless (PNG)	900 GB	168 TB
WebP 1	32 GB	5.98 TB
Ours (match, 8x)	**1.8 GB**	**0.33 TB**

Table 5: **Estimated map storage requirements using various compression methods.**

Storage Analysis: We now turn back to the approximate storage requirements described in the introduction, and showcase projected numbers when compressing all maps using our proposed method in Tab. 5. Our proposed method can compress a 5cm/px HD map of the entire Los Angeles county to just 1.8 GB, allowing it to fit in RAM on most current smartphones. We can also fit the entire USA road network at the same resolution in just 330 GB.

5. Conclusions

HD maps impose high storage requirements, which limit the ability of a self-driving fleet to operate in large-scale environments. In this paper, we proposed to learn to compress the map representation such that it is optimal for the localization task. Our experiments on a state-of-the-art localizer have shown that it is possible to learn a task-specific compression scheme that reduces storage requirements by two orders of magnitude compared to general-purpose codecs such as WebP, without sacrificing localization performance.

References

[1] JPEG - Wikipedia, 2019. 5

[2] Johannes Balle, Valero Laparra, and Eero P. Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. *2016 Picture Coding Symposium (PCS)*, 2016. 2

[3] Ioan Andrei Bârsan, Shenlong Wang, Andrei Pokrovsky, and Raquel Urtasun. Learning to localize using a lidar intensity map. In *Proceedings of The 2nd Conference on Robot Learning*, 2018. 1, 2, 3, 4, 5, 6, 7

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 6

[5] Eric Brachmann and Carsten Rother. Learning Less is More - 6D Camera Localization via 3D Surface Regression. *arXiv*, 2017. 2

[6] Georgios Floros, Benito van der Zander, and Bastian Leibe. OpenStreetSLAM: Global vehicle localization using OpenStreetMaps. In *ICRA*, 2013. 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 4

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4

[9] Ehsan Javanmardi, Mahdi Javanmardi, Yanlei Gu, and Shunsuke Kamijo. Autonomous vehicle self-localization based on probabilistic planar surface map and multi-channel LiDAR in urban area. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-March, 2018. 1, 2

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[11] Rainer Kümmerle, Dirk Hähnel, Dmitri Dolgov, Sebastian Thrun, and Wolfram Burgard. Autonomous driving in a multi-level parking structure. *ICRA*, 2009. 2

[12] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *RSS*, 2007. 1, 2

[13] Jesse Levinson and Sebastian Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *ICRA*, 2010. 1, 2, 3

[14] Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. In *CVPR*, 2016. 2

[15] Wei-Chiu Ma, Shenlong Wang, Marcus A. Brubaker, Sanja Fidler, and Raquel Urtasun. Find your way by observing the sun and other semantic cues. In *ICRA*, 2017. 2

[16] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. *arXiv preprint arXiv:1801.04260*, 2018. 2

[17] Teddy Ort, Liam Paull, and Daniela Rus. Autonomous Vehicle Navigation in Rural Environments without Detailed Prior Maps. *ICRA*, pages 1–8, 2018. 2

[18] Noha Radwan, Abhinav Valada, and Wolfram Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018. 2

[19] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *CVPR*, 2015. 2, 3

[20] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. *arXiv preprint arXiv:1705.05823*, 2017. 2

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 4

[22] Torsten Sattler, Akihiko Torii, Josef Sivic, Marc Pollefeys, Hajime Taira, Masatoshi Okutomi, and Tomas Pajdla. Are Large-Scale 3D Models Really Necessary for Accurate Visual Localization? In *CVPR*, 2017. 2

[23] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948. 5

[24] George Toderici, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*, 2015. 2, 5

[25] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *CVPR*, pages 5435–5443, 2017. 2

[26] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2276–2282. IEEE, 2006. 2

[27] Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Yao Zhou, Hao Wang, and Shiyu Song. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In *ICRA*, 2018. 2

[28] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *ICCV*, 2013. 2, 3

[29] Ryan W. Wolcott and Ryan M. Eustice. Visual localization within lidar maps for automated urban driving. In *IROS*, 2014. 1

[30] Ryan W. Wolcott and Ryan M. Eustice. Fast lidar localization using multiresolution gaussian mixture maps. In *ICRA*, 2015. 1, 2, 3

[31] Keisuke Yoneda, Hossein Tehrani, Takashi Ogawa, Naohisa Hukuyama, and Seiichi Mita. Lidar scan feature for localization with highly precise 3-d map. In *IV*, 2014. 1, 3

[32] Jure Zbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. In *CVPR*, 2015. 2