ZippyPoint: Fast Interest Point Detection, Description, and Matching through Mixed Precision Discretization

Menelaos Kanakis*,1 Simon Maurer*,1 Matteo Spallanzani[1] Ajad Chhatkuli[1] Luc Van Gool[1,2]
[1]ETH Zürich [2]KU Leuven

Abstract

Efficient detection and description of geometric regions in images is a prerequisite in visual systems for localization and mapping. Such systems still rely on traditional hand-crafted methods for efficient generation of lightweight descriptors, a common limitation of the more powerful neural network models that come with high compute and specific hardware requirements. In this paper, we focus on the adaptations required by detection and description neural networks to enable their use in computationally limited platforms such as robots, mobile, and augmented reality devices. To that end, we investigate and adapt network quantization techniques to accelerate inference and enable its use on compute limited platforms. In addition, we revisit common practices in descriptor quantization and propose the use of a binary descriptor normalization layer, enabling the generation of distinctive binary descriptors with a constant number of ones. ZippyPoint, our efficient quantized network with binary descriptors, improves the network runtime speed, the descriptor matching speed, and the 3D model size, by at least an order of magnitude when compared to full-precision counterparts. These improvements come at a minor performance degradation as evaluated on the tasks of homography estimation, visual localization, and map-free visual relocalization. Code and models are available at https://github.com/menelaoskanakis/ZippyPoint.

1. Introduction

The detection and description of geometric regions in images, such as salient points or lines, is one of the fundamental components in visual localization and mapping pipelines – essential prerequisites for Augmented Reality (AR) and robotic applications. Achieving such detection and description efficiently with handcrafted algorithms [29, 48] has produced successful robot localization methods [15, 17, 30, 37, 38]. On the other hand, Deep Neural Networks (DNNs) have significantly advanced the representational capability of descriptors by learning on large scale natural images [13], using deeper networks [14], or introducing new modules to learn feature matching [50]. However, these advances often come at the cost of more expensive models with slow run times and large memory requirements for representation storage, making them unsuitable for computationally limited platforms. While the de-

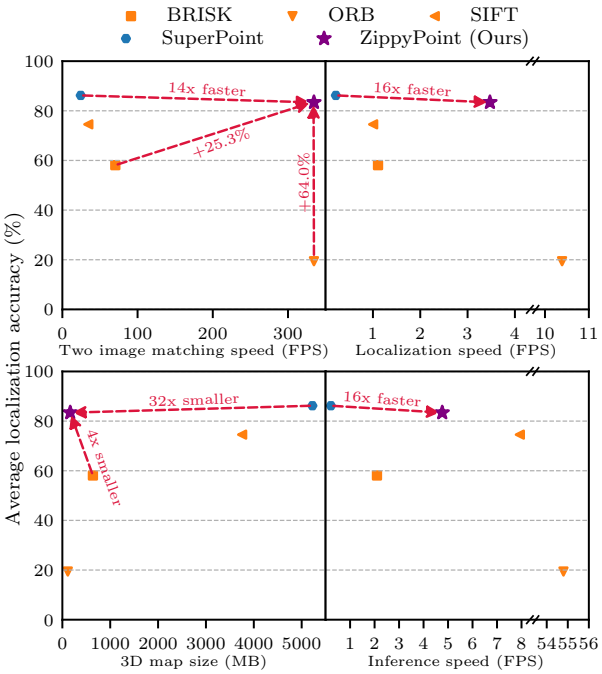


Figure 1. Learned detection and description methods, *e.g.* SuperPoint [13], significantly outperform hand-crafted methods, in orange, on challenging day-night scenarios [51, 52]. This, however, comes at the cost of slower image matching (top left), slower keypoint detection and description (bottom right), larger 3D models (bottom left), and therefore slow localization within a 3D map (top right). Speeds are reported in FPS on a CPU. In this paper we present ZippyPoint, a learned detection and description network that improves the above limitations by at least an order of magnitude while providing competitive performance, enabling its use on-board computationally limited platforms.

*M. Kanakis and S. Maurer contributed equally to this work.

mand for real-time applications such as robotics and AR is increasing, efficient DNN methods that can operate in real-time on computationally limited platforms have received surprisingly little attention.

A key component for the successful deployment of mobile robots in large-scale applications is the real-time extraction of binary descriptors. This not only enables efficient storage of the detected representation, e.g. the map in Simultaneous Localization and Mapping (SLAM) or Structure-from-Motion (SfM) pipelines, but also accelerated descriptor matching. In particular, matching computations in localization scale non-linearly with the number of images or map size. Therefore, improved two-view matching speed can translate to very high gains in real applications. Fast and light weight descriptor methods include BRISK [29], BRIEF [8] and ORB [48], however, their matching capability is often inferior to standard hand-crafted features such as SIFT [35] and SURF [5], as presented by Heinly J. *et al.* [22]. In challenging scenarios, however, hand-crafted feature extractors are outperformed significantly by learned representations [51]. While the performance gains of learned methods are highly desired, embedded platforms are limited in storage, memory, providing limited or no support for Floating-Point (FP) arithmetic, thus limiting the use of learned methods.

Motivated by the desire for improving the performance of feature points on low-compute platforms, we explore DNN quantization to enable the real-time generation of learned descriptors under such challenging constraints. However, the quantization of a DNN is not as straightforward as selecting the discretization level of convolutional layers. Quantized DNNs often require different levels of discretized precision for different layers [34, 46]. Operations such as max-pooling favour saturation regimes [46], while average pooling is affected by the required rounding and truncation operations. Moreover, prior works often focus on image-level classification tasks, with findings that do not necessarily transfer to new tasks [6]. To render the search for a Quantized Neural Network (QNN) tractable, we propose a *layer partitioning and traversal* strategy, significantly reducing the architecture search complexity. While most research considers homogeneous quantization precision across all layers [40], we find Mixed-Precision (MP) quantization yields superior performance. In addition, we find that replacing standard pooling operations with learned alternatives can further improve QNN performance.

Besides the need for real-time inference, DNNs need to additionally generate binary local descriptors for storage efficiency and fast feature matching. This adds further challenges as the discretization of the output layer draws less precise boundaries in the feature domain [28], making the network optimization more challenging. Furthermore, prior works focus on global feature description and present findings that do not trivially transfer to our task [27, 55]. To this extent, we introduce a Binary Normalization (Bin.Norm) layer that constrains the representation to a constant predefined number of ones. Bin.Norm is therefore analogous to the L_2 normalization, a staple and key component in FP metric learning [39].

In summary, our contributions are:

- We propose a heuristic algorithm, named *layer partitioning and traversal strategy*, to investigate the topological changes required for the quantization of a state-of-the-art detection and description network. We find that with a MP quantization architecture, and by replacing max-pooling with a learned alternative, we can achieve a speed-up by an order of magnitude with minor performance degradation. Our analysis reveals that the common QNN practices can be sub-optimal.

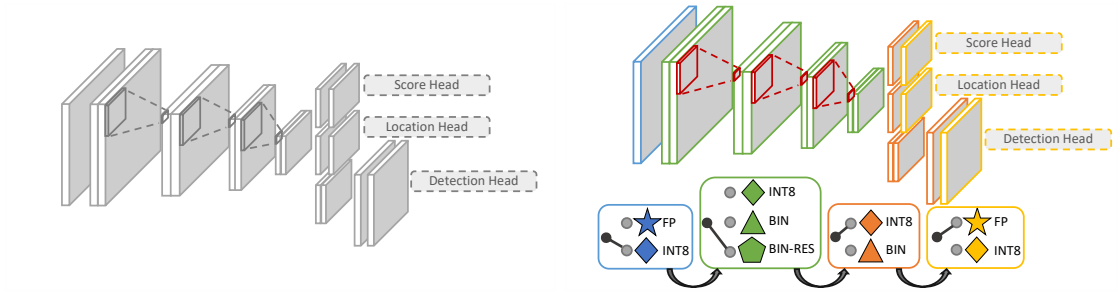
- We propose the use of a normalization layer for the end-to-end optimization of binary descriptors. Incorporating the Bin.Norm layer yields consistent improvements when compared to the common practices for descriptor binarization.

- We provide a detailed analysis of ZippyPoint, our proposed QNN with binary descriptors, on the task of homography estimation. We further demonstrate the generality of ZippyPoint on the challenging applications of Visual Localization (VisLoc) and Map-Free Visual Relocalization. ZippyPoint consistently outperforms all real-time alternatives and yields comparable performance to a full precision counterpart while addressing its known limitations, illustrated in Fig. 1.

2. Related Work

Hand-crafted feature extractors. The design of hand-crafted sparse feature extractors such as SIFT [35] and SURF [5] has been undoubtedly very successful in practice, still widely used in applications such as SfM [53]. However, the time needed for detection and descriptor extraction, coupled with their FP representation, limits them from being used on compute-limited platforms, such as light-weight unmanned aerial vehicles. Motivated by this limitation, methods like BRISK [29], BRIEF [8], and ORB [48], aimed to provide compact features targeted for real-time applications [30, 37, 38]. While fast and lightweight, they lack the representational strength to perform well under a wide variety of viewing conditions such as large viewpoint changes [22, 53] or times of day and year [51].

Learned feature extractors. Advances in DNNs have enabled the learning of robust, (pseudo-)invariant, and highly descriptive image features, pushing the boundaries of what was previously possible through hand-crafted methods.



(a) Detection and Description architecture

(b) The layer partitioning and traversal strategy

Figure 2. (a) Starting from [58], we partition the operations in macro-Blocks, depicted in (b) with different colors. From the first upstream macro-Block, in blue, we identify the optimal quantization setting that maintains functional performance while improving the network's throughput. We then traverse to the next block, green, and repeat. The strategy is complete once we have reached the most downstream network layer, the prediction heads.

While hand-crafted local features [5, 8, 29, 35, 48, 59] have not evolved much, systematic incremental progress can be seen in the learned local features [9, 13, 14, 16, 45, 47, 58]. Improvements have been achieved using contrastive learning [9], self-supervised learning [13], improved architectures [47, 58] and outlier rejection [58], to name a few approaches. Nevertheless, time and memory inefficiency remain major drawbacks of the learned methods.

In the same vein, large scale descriptor matching calls for light-weight representations. Binary descriptors enable efficient matching with moderate performance drops while significantly decreasing the storage requirements. Yet, the existing literature on binary representations focuses on image retrieval [27, 31, 42, 54, 55, 60], neglecting the detection and description of local features. For descriptor binarization, [31, 54, 55] use multistage optimization procedures. More similar to our work, [42] defines a differentiable objective for the hamming distance, [27] uses sigmoids to soften the optimization objective, while [57] rely on a hard sign function and gradient approximations. In the same spirit, we also optimize the network in a single optimization step. However, we argue that the lack of normalization layer in these methods, a staple in metric learning [39], greatly hinders the descriptor performance. To address this limitation, we propose a normalization layer for binary descriptors. Bin.Norm provides a more stable optimization process, avoids mode collapse and enables end-to-end optimization without requiring the use of gradient approximations or multiple optimization stages.

Efficient Neural Networks. Several solutions have been proposed to deploy neural networks in constrained scenarios. These solutions can be partitioned in topological optimizations, aiming at increasing accuracy-per-operation or accuracy-per-parameter [7, 21, 23], software optimizations such as tensor decomposition and parameter pruning [43, 44, 62], and hardware-aware optimizations [46].

Amongst hardware-aware optimizations, quantization plays a central role [34, 46]. By replacing FP with Integer (Int) operands, a QNN can reduce its storage and memory requirements with respect to an equivalent DNN. In addition, complex FP arithmetics can be replaced by simpler Int arithmetics. Due to these properties, QNNs can be executed at higher throughput (number of operations per cycle) and arithmetic intensity (number of arithmetic operations per memory transaction) [24]. When operand precision is extremely low, e.g. Binary (Bin), standard instruction set architectures can be exploited to increase these metrics even further [46]. Unlike mainframes and workstations, embedded platforms have limited storage and memory, limited or no support for FP arithmetics, and are optimized to execute SIMD (Single Instruction, Multiple Data) Int arithmetics. These considerations make QNNs an ideal fit for embedded applications, such as robots and mobile devices.

However, QNNs have limited representational capacity compared to their FP counterparts. Specifically, linear operations using discretized weights draw less precise boundaries in their input domains. In addition, discretized activation functions lose injectivity with respect to their FP counterparts, making quantization a lossy process [28]. To strike a balance between throughput and performance, practitioners require to identify a single Int precision [40], or alternative linear layers [34], that achieve the desired performance. These design choices are applied homogeneously across the entire network. We instead hypothesize that a single set of hyperparameters across the entire network can be suboptimal. We, therefore, investigate the use of heterogeneous layers throughout the network, e.g. different Int precision at different depths of the network, made possible through the proposed layer partitioning and traversal strategy.

3. Mixed Precision Discretization

Efficiently identifying salient points in images and encoding them with lightweight descriptors is key to enabling real-time applications such as robot localization. In this paper we explore the efficacy of learning-based descriptor methods under two constraints: minimizing run-time la-

tency and using binary descriptors for accelerating keypoint matching and efficient storage. In Sec. 3.1 we introduce the baseline architecture we initiate our investigation from. In Sec. 3.2 we propose a strategy to explore structural changes to the network's topology. In Sec. 3.3 we introduce a standard formulation of metric learning, which we use to then define our Bin.Norm descriptor layer.

3.1. Baseline Architecture

We initiate our investigation from the state-of-the-art KP2D [58] network, which exploits outlier filtering to improve detections. The KP2D model maps an input image $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ to keypoints $\mathbf{p} \in \mathbb{R}^{N\times 2}$, descriptors $\mathbf{x} \in \mathbb{R}^{N\times M}$, and keypoint scores $\mathbf{s} \in \mathbb{R}^{N}$, where N represents the total number of keypoints extracted and M the descriptor size. The model is comprised of an encoder with 4 VGG-style blocks [56], followed by a three-headed decoder for keypoints, keypoint scores, and descriptors. The encoder is comprised of 8 convolutional operations, the keypoint and keypoint score branches of 2, and the descriptor branch of 4. All convolutional operations, except for the final layers, are followed by batch normalization and leaky ReLUs [36]. The model is optimized through self-supervision by enforcing consistency in predictions between a source image $\mathbf{I}_s$ and a target image $\mathbf{I}_t = \mathrm{H}(\mathbf{I}_s)$, related through a known homography transformation $\mathbf{H}$ and its warping function H.

We chose KP2D as the starting point for our investigation due to its standard architecture design choices: a VGG style encoder [10, 13, 14, 47], encoder-decoder structure [10, 13], and the detection and description paradigm [10, 13, 14, 47]. Therefore, we expect that the investigated quantization strategy can transfer to other similar models, such as the ones listed above.

3.2. Network Quantization

For the quantization of a convolutional layer, several design choices are required. These include weight precision, feature precision, and whether to use a high precision residual. When considering independently each layer of a DNN, it leads to a combinatorially large search grid, rendering an exhaustive search of the ideal quantization policy prohibitive.

To simplify the search space, we propose the *layer partitioning and traversal strategy*, depicted in Fig. 2. First, we partition the operations of our target architecture into macro-blocks. For each macro-block, we define a collection of candidate quantized configurations. We then traverse through the macro-blocks and identify the optimal configuration for each, one at a time. This heuristic algorithm terminates once we have reached the most downstream network layer, the prediction heads. Note that, while we maintain the macro-block configurations the same once selected, the architecture is always optimized end-to-end. This strategy reduces the search complexity from combinatorial (the product of the number of configurations for each macro-block) to linear (the sum of the number of configurations for each macro-block). In addition, it ensures that when a macro-block is optimized on features with given representation capabilities, it will not degrade due to optimization of a different macro-block upstream. We detail our choice of macro-blocks and their configurations in the experiments section.

3.3. Binary Learned Descriptors

Preliminaries. When describing an image or a local region, the learned mapping aims to project a set of data points to an embedding space, where similar data are close together and dissimilar data are far apart. A fundamental component to the success of learned descriptors is the advancement of contrastive losses [12, 20, 61]. To ensure stable optimization and avoiding mode collapse, descriptors are often normalized [39]. A common selection is L_2 normalization, defined as

$$\mathbf{y} = \frac{1}{||\mathbf{x}||_2}\mathbf{x}. \tag{1}$$

While the solution, and hence gradients, can be expressed in closed-form, it assumes FP representation spaces $\mathbf{x}$ and $\mathbf{y}$. However, Bin descriptors can only take discrete values $\{0,1\}$. In search for a normalization layer applicable for Bin descriptors, we instead view and rewrite Eq. (1) as the generalized optimization objective

$$\begin{aligned} \mathbf{y} = \underset{\mathbf{z}\in\mathbb{R}^M}{arg\,\min} \quad & d(\mathbf{z};\mathbf{x}) \\ \text{subject to} \quad & constr(\mathbf{z}), \end{aligned} \tag{2}$$

where we search for the vector $\mathbf{z}$ that minimizes a distance function to $\mathbf{x}$ under a normalization constraint $constr(\mathbf{z})$. Eq. (2) is therefore equivalent to Eq. (1) when $d(\mathbf{z};\mathbf{x}) = \frac{1}{2}||\mathbf{z}-\mathbf{x}||_2^2$ and $constr(\mathbf{z})$ is $||\mathbf{z}||_2 = 1$. This enables the definition of optimization objectives that can be utilized where L_2 normalization does not provide the required behaviour. While constrained optimization problems are not differentiable, their use in DNNs is made possible through advances in deep declarative networks [19].

Normalization for Binary Descriptors. We hypothesize that normalization for binary descriptors is equivalent to having a constant number of ones in each descriptor. To this end, we take inspiration from multi-class classification problems and view the Bin.Norm as a projection of the descriptors living in an M-dimensional hypercube on a k-dimensional polytope [3]. In other words, an M-dimensional descriptor has entries that sum to k. This trivially yields the constraint from Eq. (2) to $constr(\mathbf{z}) = \mathbf{1}^\top\mathbf{z} = k$, where $\mathbf{1}$ is a vector of 1s of the same dimension as $\mathbf{z}$.

We define the new optimization objective as

$$\begin{aligned} \mathbf{y} = \underset{\mathbf{z}\in[0,1]^M}{arg\,\min} & -\mathbf{x}^\top\mathbf{z} - H(\mathbf{z}) \\ \text{subject to} \quad & \mathbf{1}^\top\mathbf{z} = k \end{aligned} \tag{3}$$

where $H(\mathbf{z})$ is the binary entropy function applied on the vector $\mathbf{z}$ for entropy based regularization.

To optimize the objective, we introduce a dual variable $\nu \in \mathbb{R}$ for the constraint of Eq. (3). The Lagrangian then becomes

$$-\mathbf{x}^T\mathbf{z} - H(\mathbf{z}) + \nu(k - \mathbf{1}^\top\mathbf{z}). \tag{4}$$

Differentiating with respect to $\mathbf{z}$, and solving for first-order optimality gives

$$-\mathbf{x} + \log\frac{\mathbf{z}^*}{\mathbf{1}-\mathbf{z}^*} - \nu^* = 0, \tag{5}$$

that yields

$$\mathbf{y} \approx \mathbf{z}^* = \sigma(\mathbf{x} + \nu^*), \tag{6}$$

where σ denotes the logistic function. We identify the optimal ν^* by using the bracketing method of [3], efficiently implemented for use on GPUs, and backpropagate using [2].

The selection of the optimization objective in Eq. (3) is two-fold. The entropy regularizer helps prevent sparsity in the gradients of the projection. In addition, the forward pass in Eq. (6) can be seen as an adaptive sigmoid that ensures the descriptor entries sum up to a specific value. This enables the direct comparison with the common practice of approximating binary entries using the sigmoid function.

At inference, the descriptor optimization strategy in Eq. 3 is replaced by a thresholding function that sets the top-k logits of each descriptor to one, for faster processing. We set the top-64 to ones, however, we found that both smaller and larger numbers yield a comparable performance.

4. Experiments

We present the implementation details in Sec. 4.1. In Sec. 4.2 we investigate the effect of network quantization using the layer partitioning and traversal strategy, as well as evaluate the proposed Bin.Norm layer for descriptor binarization. We then combine the two contributions into ZippyPoint and evaluate its performance on the task of homography estimation. We evaluate the generalization capabilities of ZippyPoint on fundamental tasks in robotic and AR pipelines, namely VisLoc in Sec. 4.3, and Map-Free Visual Relocalization in the supplementary material. We envision our work can both spark further research in the design of binary descriptors and quantized networks, as well as promote the incorporation of ZippyPoint in robotic systems.

4.1. Implementation Details

We implement our models in TensorFlow [1], and use the Larq [18] library for quantization. Our models are trained on the COCO 2017 dataset [32], comprised of 118k training images, following [10, 13, 58]. The models are optimized using ADAM [25] for 50 epochs with a batch size of 8, starting with an initial learning rate of 10^{-3} while halving it every 10 epochs. To ensure robustness in our results, we optimize each model configuration three times and report the mean and standard deviation.

To enable self-supervised training, spatial and non-spatial augmentations for the homography transformation are required. For spatial transformations, we utilize crop, translation, scale, rotation, and symmetric perspective. Non-spatial transformations applied are per-pixel Gaussian noise, Gaussian blur, color augmentation in brightness, contrast, saturation, and hue. Finally, we randomly shuffle the color channels and convert images to gray scale. Please refer to [58] for more details.

4.2. Designing ZippyPoint

We conduct our DNN quantization investigation on the task of homography estimation, a commonly used task for the evaluation of self-supervised learned models [10,13,58]. Homographic transformations largely eliminates domain shifts due to missing 3D, providing a good benchmark for ablation studies.

We evaluate our method on image sequences from the HPatches dataset [4]. HPatches contains 116 scenes, separated in 57 illumination and 59 viewpoint sequences. Each sequence is comprised of 6 images, with the first image used as a reference. The remaining images are used to form pairs for evaluation. As is common practice, we report Repeatability (Repeat.), Localization Error (Loc), Matching Score (M.Score), and Homography Accuracy with thresholds of 1, 3 and 5 pixels (Cor-1, Cor-3, Cor-5). We additionally benchmark and report the CPU speeds in Frames Per Seconds (FPS) on an Apple M1 ARM processor.

Baseline. We initiate our investigation in Table 1 from a re-implementation of KP2D with minor modifications to enable a structured search with minimal macro-block interference. Specifically, KP2D uses a shortcut connection between the encoder and decoder macro-blocks. We remove this skip-connection to constrain the interaction between two macro-blocks to a single point. Furthermore, we replace the leaky ReLUs with hard-swish [23], a comparable but faster alternative. The functional performance of *Baseline* is comparable to KP2D while slightly improving the throughput.

We then partition our baseline architecture into macro-blocks. These include the first encoder convolution, the remaining encoder convolutions, spatial reduction layers, the

Table 1. Results from the layer partitioning and traversal strategy. The final model, in bold, performs comparably to the baseline while running an order of magnitude faster. Green highlights the configuration used in the next stage.

						Repeat. ↑	Loc. ↓	Cor-1 ↑	Cor-3 ↑	Cor-5 ↑	M.Score ↑	FPS ↑
KP2D [58]						0.686	0.890	0.591	0.867	0.912	0.544	2.5
Baseline (ours)						0.649 ± 0.004	0.792 ± 0.015	0.566 ± 0.015	0.880 ± 0.011	0.925 ± 0.007	0.571 ± 0.004	3.6
Encoder Convolutions												
First conv		Remaining convs										
FP	Int8	Int8	Bin	Bin-R								
✓		✓				0.651 ± 0.004	0.840 ± 0.076	0.528 ± 0.019	0.867 ± 0.017	0.922 ± 0.011	0.574 ± 0.004	10.6
	✓	✓				0.657 ± 0.003	0.825 ± 0.015	0.548 ± 0.003	0.866 ± 0.004	0.925 ± 0.010	0.577 ± 0.001	13.8
	✓		✓			0.561 ± 0.036	1.120 ± 0.061	0.281 ± 0.132	0.401 ± 0.022	0.449 ± 0.080	0.247 ± 0.096	15.2
	✓			✓		0.653 ± 0.005	1.040 ± 0.018	0.401 ± 0.034	0.811 ± 0.011	0.890 ± 0.007	0.563 ± 0.006	14.5
Spatial Reduction												
Max	Aver.	Sub.S.	Learn	E.Learn								
✓						0.653 ± 0.005	1.040 ± 0.018	0.401 ± 0.034	0.811 ± 0.011	0.890 ± 0.007	0.563 ± 0.006	14.5
	✓					0.656 ± 0.003	1.068 ± 0.033	0.354 ± 0.038	0.788 ± 0.027	0.873 ± 0.011	0.558 ± 0.015	13.9
		✓				0.640 ± 0.022	1.128 ± 0.053	0.362 ± 0.024	0.783 ± 0.003	0.881 ± 0.007	0.537 ± 0.015	14.4
			✓			0.648 ± 0.009	0.890 ± 0.066	0.517 ± 0.029	0.848 ± 0.009	0.916 ± 0.003	0.571 ± 0.006	14.2
				✓		0.656 ± 0.002	0.943 ± 0.034	0.491 ± 0.024	0.844 ± 0.015	0.906 ± 0.005	0.568 ± 0.002	16.2
Decoder Convolutions												
Remaining convs			Descriptor conv									
Int8	BIN-R		FP	Int8								
✓			✓			0.658 ± 0.002	0.964 ± 0.020	0.488 ± 0.030	0.840 ± 0.013	0.900 ± 0.001	0.569 ± 0.002	27.2
	✓		✓			0.655 ± 0.003	1.018 ± 0.006	0.451 ± 0.001	0.456 ± 0.001	0.481 ± 0.002	0.329 ± 0.008	30.1
✓				✓		**0.652 ± 0.005**	**0.926 ± 0.022**	**0.506 ± 0.025**	**0.853 ± 0.007**	**0.917 ± 0.003**	**0.571 ± 0.003**	**47.2**

non-head decoder convolutions, and the head decoder convolutions, as depicted in Fig. 2 by the different colours.

Macro-Block I: First Encoder Convolution. For macro-block I, we considered two configurations: FP and Int8. Although [34, 46] suggest that keeping the first convolution in FP has a negligible effect on application throughput while degradation of functional performance, our findings suggest otherwise. Specifically, we find that using an Int8 convolution improves throughput by as much as 3 FPS, while having no detectable impact on functional performance. We ascribe this to the fact that the input images are also represented in Int8. Therefore, discretization of the input sequence does not cause a loss of information, while enabling the use of a more efficient Int8 convolution.

Macro-Block II: Encoder Convolutions. For the encoder convolutions, we considered three configurations: Int8, Bin, and Binary with a high-precision Residual (Bin-R). While using Bin convolutions in the encoder significantly improves application throughput, functional performance is severely hindered, as measured by the halving of the correctness metrics. This drop is consistent with findings in the literature for semantic segmentation [63] while conflicting with image-level classification experiments [46]. This further supports our arguments for the importance of task-specific investigations.

To alleviate such drastic performance drops, we introduce high precision Int8 representations in the form of a residual operation. For convolutional operations with a mismatch in the number of input and output channels, we introduce additional Int8 1×1 convolutions on the residual path. This ensures the high-precision paths maintain their Int8 precision, while matching the channel dimensions. The additional high-precision Int8 residuals improve performance significantly. This again advocates for the redundancy of FP representation in the encoder, as the encoder is now bottlenecked by Int8 precision.

Macro-Block III: Spatial Reduction. For the spatial reduction layers, we considered four configurations: max-pooling (Max), average-pooling (Aver.), sub-sampling (Sub.S.) and a learned projection (Learn). As is common in DNNs, our baseline utilizes max-pooling. However, max-pooling has been found to favour saturated regimes and therefore eliminates information when applied on low-precision features like those found in QNNs [46]. Average pooling further degrades the performance, attributed to the errors introduced due to the roundings and truncations which are essential for integerized arrays. To further highlight this error, a simple Sub.S. that only uses information from a quarter of the kernel window yields comparable performance to Aver.

To alleviate the challenges highlighted above, we propose the use of a learned pooling operation (Learn). The learned pooling comes in the form of an Int8 convolutional operation with the same kernel size and stride as the other pooling operations. We select Int8 so as to maintain the representational precision of the network, defined by the macro-block I and the high precision residuals. While operating at a comparable run-time to max pooling, the performance significantly improved. This further corroborates our hypothesis that learned pooling can address both the aforementioned challenges. Finally, we investigate the effect of the pooling placement (E.Learn). Specifically, we change the location of the pooling operations from the end to the beginning of each convolutional block. While with

Table 2. We evaluate the efficacy of different normalization layers when combined with using sigmoid as a soft approximation for every bit. We find that the binary normalization (Bin.Norm) layer for the descriptors consistently improves all metrics.

	Norm	Repeat. ↑	Loc. ↓	Cor-1 ↑	Cor-3 ↑	Cor-5 ↑	M.Score ↑
Full Precision	L_2	0.644 ± 0.003	0.788 ± 0.005	0.580 ± 0.007	0.886 ± 0.008	0.933 ± 0.010	0.569 ± 0.003
Sigmoid		0.640 ± 0.005	0.809 ± 0.049	0.173 ± 0.300	0.285 ± 0.493	0.305 ± 0.528	0.187 ± 0.318
Sigmoid + $\|\|L_2\|\|$	L_2	0.650 ± 0.001	0.803 ± 0.010	0.491 ± 0.015	0.822 ± 0.009	0.888 ± 0.003	0.513 ± 0.003
Sigmoid + Bin.Norm (Ours)	Bin.Norm	**0.651 ± 0.003**	**0.796 ± 0.016**	**0.545 ± 0.005**	**0.880 ± 0.090**	**0.925 ± 0.004**	**0.553 ± 0.002**

Table 3. We compare ZippyPoint with full precision or binary descriptors against state-of-the-art methods. ZippyPoint performs on par with other full-precision methods while running an order of magnitude faster than the full-precision alternative. When compared to binary hand-crafted methods, ZippyPoint consistently outperforms all other methods, often by a large margin.

	Repeat. ↑	Loc. ↓	Cor-1 ↑	Cor-3 ↑	Cor-5 ↑	M.Score ↑
			Full-Precision Descriptors			
SuperPoint [13]	0.631	1.109	0.491	0.833	0.893	0.318
SIFT [35]	0.451	**0.855**	**0.622**	0.845	0.878	0.304
SURF [5]	0.491	1.150	0.397	0.702	0.762	0.255
KP2D [58]	**0.686**	0.890	0.591	**0.867**	0.912	0.544
ZippyPoint (Ours)	0.652 ± 0.005	0.926 ± 0.022	0.506 ± 0.025	0.853 ± 0.007	**0.917 ± 0.003**	**0.571 ± 0.003**
			Binary Descriptors			
BRISK [29]	0.566	1.077	0.414	0.767	0.826	0.258
ORB [48]	0.532	1.429	0.131	0.422	0.540	0.218
ZippyPoint (Ours)	**0.652 ± 0.005**	**0.926 ± 0.022**	**0.433 ± 0.007**	**0.820 ± 0.007**	**0.887 ± 0.006**	**0.571 ± 0.003**

FP convolutional layers this would cause a $4\times$ speedup for each convolution, in quantized convolutions the gain is even greater [18].

Macro-Block IV: Decoder Convolutions. For the decoder convolutions, we considered two configurations: Int8, and Bin-R. We do not investigate Bin due to the large performance drop observed in macro-block II. Unlike the findings from macro-block II, our decoder experiments demonstrate the importance of Int8, highlighting the benefits of MP networks. Specifically, utilizing Int8 for the entire network would not yield the best throughput, as seen in macro-block II, while Bin-R for the entire network would not yield the best performance.

Macro-Block V: Final Decoder Convolutions. For the final convolutions, we evaluated FP and Int8 for the score, location and descriptor heads independently. We find that score and location heads require FP representations, with models often failing to optimize otherwise. On the other hand, the descriptor branch can be optimized with Int8, significantly improving the throughput.

Network Quantization Findings. Network latency can be significantly improved when quantizing the first convolutional layer and the last descriptor head to Int8, while having an insignificant effect on functional performance. This is contrary to findings from prior works [34, 46]. In addition, enhanced performance can be achieved through MP QNNs. In other words, a network comprised of only Bin-R or Int8 convolutional operations would yield sub-optimal results. This observation suggests that good quality and general-purpose features can be extracted using low-precision convolutions when coupled with higher precision residuals. Furthermore, it suggests that the dense task predictor heads benefit from higher Int8 precision to accurately reconstruct the target information from the encoded features. Finally, we observe that the prediction heads for regression tasks (score and location) cannot be quantized and should be left in FP, while the descriptor head can be quantized to Int8. This further drives the importance of the structured investigation, like the layer partitioning and traversal strategy.

Binarizing descriptors. We initiate the descriptor exploration from the common practice of utilizing sigmoid as a soft approximation for every bit [27, 33], and the hamming triplet loss proposed by [27]. While some works use a hard sign function [57], we found it unable to optimizate the network to a meaningful degree. Table 2 demonstrates a significant performance drop and large variance compared to the baseline, especially in the correctness metrics. We conjecture that this spans from the lack of a normalization layer, causing the sigmoid to saturate, and yielding uninformative gradients. To test this assumption, we append an $||L_2||$ normalization layer after the element-wise sigmoid operation. This constrains the activations and dramatically improves performance and reduces the variance, as seen experimentally, leading to a more stable optimization process.

In this paper we hypothesize that, analogous to $||L_2||$ normalization, we can optimize the network using a Bin normalization layer by constraining the descriptor to a constant number of ones. Using the proposed Bin.Norm layer, the functional performance gap is significantly decreased when compared to the FP descriptors.

Comparison to state-of-the-art. We compare ZippyPoint with state-of-the-art methods in Table 3. For a fair com-

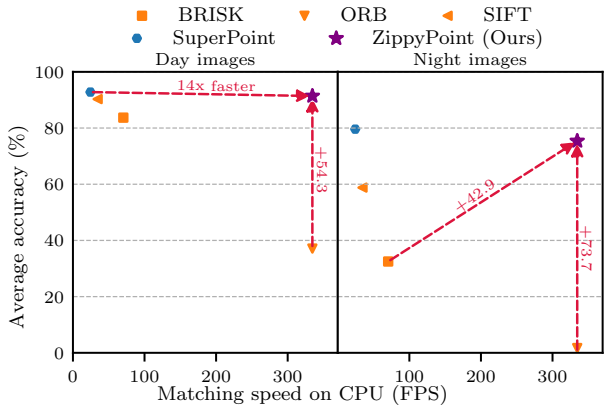


Figure 3. Comparison of the average visual localization accuracy vs descriptor matching speed between two images on the AachenV1.1 Day-Night datasets. ZippyPoint consistently outperforms all other binary methods.

parison, we group methods given the descriptor precision. When utilizing FP descriptors, ZippyPoint performs on par with other methods. In particular, it consistently outperforms SuperPoint and performs on par with KP2D. Meanwhile, the throughput gain is higher than an order of magnitude.

The benefits of ZippyPoint, the combination of the fast QNN architecture from Table 1 and the binary optimization strategy from Table 2, become apparent when comparing binary descriptor methods. We consistently outperform ORB [37] by a large margin in all metrics. We additionally outperform BRISK [29] in all metrics and even report double the matching score, a crucial metric for adaptation of these methods in downstream tasks like VisLoc. Here on, we refer to ZippyPoint as our QNN with binary descriptors.

4.3. Visual Localization

Camera localization is one of the key components in several robotic and mapping applications. Both relative [41] and absolute [26] camera localization require good local feature point descriptors to match, and are key building blocks in seminal pipelines [11, 15, 17, 30, 37, 38]. To further demonstrate the potential of ZippyPoint, we assess its generalization capability on the task of absolute camera localization, where the pose of a query image is estimated with respect to a 3D map.

We utilize the hloc framework [49], similar to prior works [47, 50], and evaluate the performance on the challenging real-life AachenV1.1 Day-Night datasets from the VisLoc benchmark [51, 52]. More precisely, we reconstruct the 3D map using ZippyPoint features instead of SIFT [35]. For each query image, we perform a coarse search of the map and retrieve the 30 closest database images based on their global descriptors, representing candidate locations. The query image is then localized within the 3D map by utilizing the candidate locations. Please refer to [49] for more details.

The results are presented in Fig. 3 with respect to the FPS speed for matching two images. In Fig. 1 we additionally depict the average performance score for both day and night query sets with respect to 3D model size, query localization time, and model inference speed. While ZippyPoint performs comparably to SuperPoint during day time, we decrease the 3D model size, query localization time, and model inference speed by at least an order of magnitude. This is attributed to the lightweight binary descriptors, the more efficient similarity comparison between the descriptors, and the network quantization. Localization with ZippyPoint at night is slightly inferior to SuperPoint, however, we expect optimization of the image transformations during training can close this gap further.

On the binary descriptor front, ZippyPoint consistently outperforms ORB by a significant margin at a comparable matching speed. BRISK on the other hand is competitive to ours on the day dataset, with the slower run-time of BRISK attributed partly to the larger descriptor size, twice that of ZippyPoint, and the increased number of detected keypoints. However, the more challenging night dataset paints a different picture, with ZippyPoint outperforming BRISK by 42.9% and ORB failing to localize. This further attests to the need for efficient learned detection and description networks, in particular for more challenging and adverse conditions.

5. Conclusion

In this paper, we investigated efficient detection and description of learned local image points through mixed-precision quantization of network components and binarization of descriptors. To that end, we followed a structured investigation, we refer to as layer partitioning and traversal for the quantization of the network. In addition, we proposed the use of a binary normalization layer to generate binary descriptors with a constant number of ones.

We obtained an order of magnitude throughput improvement with minor degradation of performance. In addition, we find that the binary normalization layer allows the network to operate on par with full-precision networks, while consistently outperforming hand-crafted binary descriptor methods. The results show the suitability of our approach on visual localization and map-free visual relocalization, challenging downstream tasks and essential prerequisites for robotic applications, while significantly decreasing the 3D model size, matching, and localization speed. We believe ZippyPoint can spark further research towards bringing learned binary descriptor methods to mobile platforms, as well as promote its incorporation in both new and established robotic pipelines.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5

[2] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017. 5

[3] Brandon Amos, Vladlen Koltun, and J Zico Kolter. The limited multi-label projection layer. *arXiv*, 2019. 4, 5

[4] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *CVPR*, 2017. 5

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *ECCV*, 2006. 2, 3, 7

[6] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Back to simplicity: How to train accurate bnns from scratch? *arXiv*, 2019. 2

[7] Philippe Blatter, Menelaos Kanakis, Martin Danelljan, and Luc Van Gool. Efficient visual tracking with exemplar transformers. In *WACV*, 2023. 3

[8] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *ECCV*, 2010. 2, 3

[9] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *NIPS*, 2016. 3

[10] Peter Hviid Christiansen, Mikkel Fly Kragh, Yury Brodskiy, and Henrik Karstoft. Unsuperpoint: End-to-end unsupervised interest point detector and descriptor. *arXiv*, 2019. 4, 5

[11] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *T-PAMI*, 29(6):1052–1067, 2007. 8

[12] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019. 4

[13] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPRW*, 2018. 1, 3, 4, 5, 7

[14] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-net: A trainable cnn for joint description and detection of local features. In *CVPR*, 2019. 1, 3, 4

[15] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *ICRA*, 2012. 1, 8

[16] Mohammed E Fathy, Quoc-Huy Tran, M Zeeshan Zia, Paul Vernaza, and Manmohan Chandraker. Hierarchical metric learning and matching for 2d and 3d geometric correspondences. In *ECCV*, 2018. 3

[17] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *T-RO*, 28(5):1188–1197, 2012. 1, 8

[18] Lukas Geiger and Plumerai Team. Larq: An open-source library for training binarized neural networks. *Journal of Open Source Software*, 5(45):1746, Jan. 2020. 5, 7

[19] Stephen Gould, Richard Hartley, and Dylan John Campbell. Deep declarative networks. *T-PAMI*, 2021. 4

[20] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006. 4

[21] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015. 3

[22] Jared Heinly, Enrique Dunn, and Jan-Michael Frahm. Comparative evaluation of binary features. In *ECCV*, 2012. 2

[23] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 3, 5

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018. 3

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[26] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR*, 2011. 8

[27] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 2015. 2, 3, 7

[28] Gian Paolo Leonardi and Matteo Spallanzani. Analytical aspects of non-differentiable neural networks. *arXiv*, 2020. 2, 3

[29] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *ICCV*, 2011. 1, 2, 3, 7, 8

[30] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *IJRR*, 34(3):314–334, 2015. 1, 2, 8

[31] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *CVPR*, 2016. 3

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 5

[33] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, 2012. 7

[34] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018. 2, 3, 6, 7

[35] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 2, 3, 7, 8

[36] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. 4

[37] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE T-RO*, 31(5):1147–1163, 2015. 1, 2, 8

[38] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *T-RO*, 33(5):1255–1262, 2017. 1, 2, 8

[39] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *ECCV*, 2020. 2, 3, 4

[40] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *ICCV*, 2019. 2, 3

[41] David Nistér. An efficient solution to the five-point relative pose problem. *T-PAMI*, 26(6):756–770, 2004. 8

[42] Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. Hamming distance metric learning. *NIPS*, 2012. 3

[43] Anton Obukhov, Maxim Rakhuba, Stamatios Georgoulis, Menelaos Kanakis, Dengxin Dai, and Luc Van Gool. T-basis: a compact representation for neural networks. In *ICML*, 2020. 3

[44] Anton Obukhov, Maxim Rakhuba, Alexander Liniger, Zhiwu Huang, Stamatios Georgoulis, Dengxin Dai, and Luc Van Gool. Spectral tensor train parameterization of deep learning layers. In *AISTATS*, 2021. 3

[45] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: Learning local features from images. *NeurIPS*, 2018. 3

[46] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 2, 3, 6, 7

[47] Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2d2: repeatable and reliable detector and descriptor. In *NeurIPS*, 2019. 3, 4, 8

[48] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, 2011. 1, 2, 3, 7

[49] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019. 8

[50] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 1, 8

[51] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, et al. Benchmarking 6dof outdoor visual localization in changing conditions. In *CVPR*, 2018. 1, 2, 8

[52] Torsten Sattler, Tobias Weyand, Bastian Leibe, and Leif Kobbelt. Image retrieval for image-based localization revisited. In *BMVC*, 2012. 1, 8

[53] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 2

[54] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR*, 2015. 3

[55] Fumin Shen, Yan Xu, Li Liu, Yang Yang, Zi Huang, and Heng Tao Shen. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *T-PAMI*, 40(12):3034–3044, 2018. 2, 3

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014. 4

[57] Jiexiong Tang, Ludvig Ericson, John Folkesson, and Patric Jensfelt. Gcnv2: Efficient correspondence prediction for real-time slam. *RA-L*, 4(4):3505–3512, 2019. 3, 7

[58] Jiexiong Tang, Hanme Kim, Vitor Guizilini, Sudeep Pillai, and Rares Ambrus. Neural outlier rejection for self-supervised keypoint learning. In *ICLR*, 2020. 3, 4, 5, 6, 7

[59] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *T-PAMI*, 32(5):815–830, 2009. 3

[60] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *T-PAMI*, 40(4):769–790, 2017. 3

[61] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10(2), 2009. 4

[62] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *T-PAMI*, 38(10):1943–1955, 2015. 3

[63] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In *CVPR*, 2019. 6