Binary Neural Networks for Memory-Efficient and Effective Visual Place Recognition in Changing Environments

Bruno Ferrarini, Michael J. Milford, *Senior Member, IEEE*, Klaus D. McDonald-Maier, *Senior Member, IEEE*, and Shoaib Ehsan, *Senior Member, IEEE*

***Abstract*—Visual place recognition (VPR) is a robot's ability to determine whether a place was visited before using visual data. While conventional handcrafted methods for VPR fail under extreme environmental appearance changes, those based on convolutional neural networks (CNNs) achieve state-of-the-art performance but result in heavy runtime processes and model sizes that demand a large amount of memory. Hence, CNN-based approaches are unsuitable for resource-constrained platforms, such as small robots and drones. In this article, we take a multistep approach of decreasing the precision of model parameters, combining it with network depth reduction and fewer neurons in the classifier stage to propose a new class of highly compact models that drastically reduces the memory requirements and computational effort while maintaining state-of-the-art VPR performance. To the best of our knowledge, this is the first attempt to propose binary neural networks for solving the VPR problem effectively under changing conditions and with significantly reduced resource requirements. Our best-performing binary neural network, dubbed FloppyNet, achieves comparable VPR performance when considered against its full-precision and deeper counterparts while consuming 99% less memory and increasing the inference speed by seven times.**

***Index Terms*—Binary neural networks, localization, visual-based navigation.**

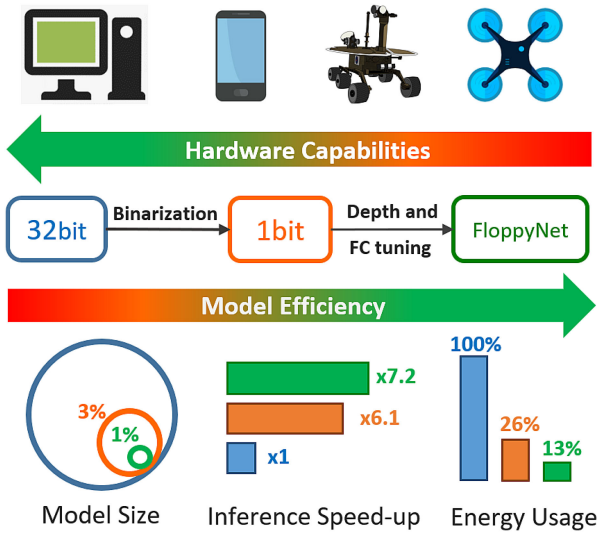

Fig. 1. FloppyNet is a compact and efficient binary network designed to enable VPR on edge devices and robots with severe hardware constraints.

I. Introduction

VISUAL place recognition (VPR) addresses the problem of determining whether a location has been visited before using visual information. VPR is a fundamental task for autonomous navigation. It enables a robot to relocalize itself in the workspace when the position tracking fails or drifts due to accumulated errors. However, variations in viewpoint and appearance due to seasonal, weather, and illumination changes render VPR challenging for mobile robots. While conventional handcrafted techniques for VPR fail under extreme environmental changes, those based on deep convolutional neural networks (CNNs) achieve state-of-the-art performance [81] but result in heavy runtime processes and model sizes that demand a large amount of memory.

Mobile robots are often equipped with resource-constrained hardware that limits the usability of such demanding techniques [32], [55]. Increasing the efficiency by saving memory and reducing the computational effort to run a model without sacrificing the performance is paramount for such resource-constrained mobile robots. Higher efficiency enables VPR on cheap hardware and frees resources for additional functionalities to improve a robot's navigation system.

Reducing resource demand while keeping VPR performance at a reasonable level is a difficult task. To tackle this challenge, in this article, we propose the multistep approach summarized in Fig. 1 that combines binary neural networks (BNNs) [26], [41] and depth reduction to obtain very compact models that

Manuscript received June 20, 2021; revised November 28, 2021; accepted January 20, 2022. This paper was recommended for publication by Associate Editor J. Civera and Editor F. Chaumette upon evaluation of the reviewer's comments. This work was supported by the U.K. Engineering and Physical Sciences Research Council under Grant EP/R02572X/1 and Grant EP/P017487/1, and in part by the RICE Project funded by the National Centre for Nuclear Robotics Flexible Partnership Fund. The work of M. Milford was supported by the QUT Centre for Robotics. *(Corresponding author: Bruno Ferrarini.)*

Bruno Ferrarini, Klaus D. McDonald-Maier, and Shoaib Ehsan are with the School of Computer Science and Electronic Engineering, University of Essex, CO4 3SQ Colchester, U.K. (e-mail: bferra@essex.ac.uk; kdm@essex.ac.uk; sehsan@essex.ac.uk).

Michael J. Milford is with the QUT Centre for Robotics, School of Electrical Engineering and Robotics, Brisbane, QLD 4000, Australia (e-mail: michael.milford@qut.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2022.3148908.

Digital Object Identifier 10.1109/TRO.2022.3148908

drastically decrease the memory requirements and improve computational efficiency. The subsequent VPR performance loss is mostly countered by training the model with a classifier stage, including a reduced number of full-precision neurons.

BNNs are a class of networks characterized by a single-bit precision for both weights and activations instead of the 32 bits used by conventional deep neural networks. So far, BNNs have been employed and highly optimized for classification tasks only, where they exhibit lower yet comparable accuracy to their full-precision counterparts [12], [68]. However, classification and VPR are different problems. The first aims to find the best fit among categories, while VPR consists of matching different images of the same scene. To the best of our knowledge, this article is the first attempt to employ BNNs to solve the VPR problem effectively under environmental changes and with significantly reduced memory requirements and computational effort. Our best model,[1] dubbed FloppyNet, achieves comparable VPR performance to its full-precision and deeper counterparts while consuming 99% less memory and running seven times faster. With a model size of 154 kilobytes, FloppyNet can therefore be stored in an old $5^{1/4}$-inch floppy disk!

The rest of this article is organized as follows. Section II presents an overview of the related work. Section III presents the proposed multistep approach to obtain compact BNNs for VPR. The evaluation criteria and the experimental setup are described in Section IV. A comprehensive binary layers analysis for VPR applications is proposed in Section V. VPR performance results and real-time benchmarks are presented and discussed in Sections VI and VII, respectively. Section VIII compares the proposed BNN with several handcrafted image descriptors. Finally, Section IX concludes this article.

II. Related Work

A. Visual Place Recognition

Environmental changes, such as illumination and viewpoint variations, render VPR a very challenging task. As the core problem of VPR is image matching, computing a robust image representation is fundamental for developing reliable localization systems in dynamic environments.

Before CNNs became popular for computer vision applications, the techniques employed for image matching consisted mainly of handcrafted image descriptors. Histogram-of-oriented gradients (HOG) [29], [34] calculates the gradient of every image pixel to create a histogram, with each bar representing a gradient angle and magnitude. HOG can be either used for VPR as a global image descriptor [57] or to describe regions of interest in an image [79]. SIFT [52] detects key-points in an image using difference-of-Gaussian (DoG) and uses HOG to compute a descriptor of their neighborhood. SURF [16] is partially inspired by SIFT. It is employed in a variety of VPR methods [28], [60]. Gist [61] is used for image matching in [72] and [70]. Gist employs a set of Gabor filters at different orientations and frequencies to extract global features from an image. Those features are then averaged and combined into a vector representing the image. CoHOG [79] is a recent image descriptor proposed as a trainingless and computationally efficient alternative to CNN-based techniques. It uses image-entropy [80] to detect regions of interest that are subsequently assigned with a HOG descriptor. CoHOG is designed to achieve lateral viewpoint tolerance.

In recent years, machine learning techniques have become more and more popular in VPR applications. CNN-based methods achieve high performance in various environmental conditions [83] and under viewpoint variations [82]. A pretrained CNN for a different task can be used off-the-shelf for generating an image descriptor in place of handcrafted local and global image descriptors. For example, the features computed by the convolutional layers of AlexNet [48] can be used to match place images. Hou *et al.* [39] showed that the features extracted from *conv3* layer of AlexNet are robust to condition variations, while those from *pool5* work well for viewpoint changes. Bai *et al.* [14] used those layers' features to improve the matching performance of SeqSLAM [59] under viewpoint changes. AMOSNet and HybridNet [23] are variants of AlexNet trained on Specific PlacEs Dataset (SPED) [23] in order to compute more specific image representations for VPR. PlaceNet [84] is based on the same idea, but it uses VGG-16 [69], which is trained on a large dataset, dubbed Places365, organized in 365 place categories. CALC [58] is a lightweight CNN proposed for addressing the loop closure detection problem efficiently. CALC is trained using an autoencoder to recreate a HOG descriptor from geometrically distorted place images. Cross-Region-Bow [24], the regional maximum activation of convolutions (R-MAC) [74], CAMAL [46], and Region-VLAD [45] focus on features pooling from a pretrained network as they consider feature extraction and aggregation as two separated stages. On the other hand, NetVLAD [13] consists of two stages trained end-to-end. The first is a VGG-16 network that extracts the features from an image followed by an aggregation layer to combine them in a VLAD-like descriptor [43].

B. Binary Neural Networks

While CNNs are effective in addressing VPR, they include many parameters that result in a large model size and heavy computational effort. In the last decade or so, several techniques have been proposed to decrease models' runtime requirements. Early approaches targeted redundant and noninformative weights: Optimal Brain Damage [50] and Optimal Brain Surgeon [38] decrease the number of connections using the Hessian of the loss function. Han *et al.* [37] showed how to reduce the number of parameters by one order of magnitude in several state-of-the-art networks by weight pruning. Also, a network's size can be shrunk by lowering the precision of the weights. However, postquantization yields performance loss, which is more prominent as the precision lowers. In particular, posttraining binarization (1-bit precision) enables the highest model compression and computational speedup but impacts heavily on a classifier's accuracy [27].

Binary-aware training enables low-precision models with acceptable classification accuracy [68]. Although training

[1] [Online]. Available: https://github.com/bferrarini/FloppyNet_TRO

binary models from scratch was attempted decades ago [66], only recently, gradient-based techniques have become applicable to BNNs. Courbariaux and Bengio [26] trained a full binary network for the first time using straight-through-estimator (STE) [17]. The key idea of STE is to keep in memory real-valued weights, which are binarized only in the forward pass to compute neurons' activation and updated during backpropagation as in a standard neural network. Afterward, several additions to the field were proposed to improve BNNs. In XNOR-Net [65], the convolutional blocks (CBs) are rearranged to increase classification accuracy. Batch-Normalization (BatchNorm) is usually placed after the convolution and before the activation function. In XNOR-Net, BatchNorm and binary activation precede convolution so that pooling occurs before binarization. DoReFa-Net [85] exploits bitwise operations to compute the dot product between a layer's weights and the inputs in an efficient way to speed up training. In [31], binarization threshold is learned along with the weights to shorten the accuracy gap with full-precision classifiers. Networks using a less extreme quantization have been proposed as a more accurate alternative to binary networks. Ternary networks [51], [86] use three values to encode weights. Although they exhibit a significant memory reduction and simple arithmetics, ternary networks require 2-bits to store weights and do not outperform BNNs by a wide margin [68].

To the best of our knowledge, BNNs have been used only for classification so far. Unlike regular CNNs, BNNs have not been considered for VPR yet. This article aims to contribute to the field by proposing a class of highly compact binary networks to solve the VPR problem effectively in changing environments.

C. BNNs Inference Frameworks

Binarization reduces a model's size dramatically and enables efficient convolution computation. However, BNNs cannot express their full potential without an inference engine that can efficiently compute bitwise operations. Different hardware platforms implement binary primitives, such as XNOR and popcount, differently. Therefore, inference libraries typically target one or a few hardware platforms to guarantee that the deployed models can run efficiently. In this section, we present a selection of the most relevant libraries and tools for deploying BNNs.

Courbariaux and Bengio [26] released a GPU kernel-based that speeds up convolutions by seven times using SIMD (single instruction, multiple data) within a register (SWAR) technique. DaBNN [20] is a stand-alone library to deploy BNNs on ARM platforms. Binary convolutions are computed by combining im2col transformation and GEMM (GEneral Matrix Multiplication). DaBNN uses *ad hoc* implementation written in ARM assembly that enables $8-10\times$ speedup compared to a full-precision implementation. BMXNet [87] extends MXNet [21] providing a complete ecosystem to train and deploy BNNs. Like DaBNN, it computes binary convolutions by combining im2col with GEMM. BMXNet utilizes standard C++ to implement binary operations reaching a $13\times$ speedup compared to floating-point convolutions written with the CBLAS library [77]. Riptide [35] is another end-to-end framework for training and deploying BNNs, focusing on integrating binary convolutions with those layers that cannot be binarized, such as BatchNorm and activation functions. Models are trained with Tensorflow [11] and compiled for deployment with TVM [22]. A binary model compiled with Riptide is $4\times$ to $12\times$ faster than its full-precision counterpart. Larq Compute Engine (LCE) [15] is a part of the Larq project [36] to train and deploy BNNs. As Riptide, LCE aims to integrate efficiently binary convolutions with a model's full-precision components. Also, LCE proposes a binary kernel highly optimized for ARM platforms. The convolution speedup is $8.5-18.5\times$. Finally, FINN [76] is an experimental tool targeting FPGAs supporting PYNQ [7]. FINN does not include a training framework but can compile PyTorch [64] models optimized with Bravitas [63].

D. Deep Neural Networks Benchmark Analysis

Benchmark analysis is an important step to take for understating a model's usability with the target hardware. The dominant evaluation metrics are inference time (or inference latency), power consumption, and memory usage. Those metrics are particularly relevant for robotic applications as they often rely on constrained hardware and battery supply. TANGO [44] employs those metrics to assess CNN models deployed on several hardware platforms. The importance of energy usage is emphasized by Palit *et al.* [62], who present an energy estimation model along with empirical data from well-established CNNs. DNNTune [78] uses inference time and energy consumption to tune both CNNs and quantized networks for several application scenarios.

Finding the best possible trade-off between memory usage and performance is an essential task when resource-constrained hardware is employed with deep neural networks. Howard *et al.* [40] proposed a class of efficient networks for mobile applications using the classification accuracy to parameters number as a tuning criterion. Bianco *et al.* [19] employ the *accuracy density* metric to represent the efficiency of deployed models at using their parameters. Unlike CNN, BNN's memory allocation efficiency received little or no attention so far. BNNs' memory footprint is usually evaluated only relatively to full-precision counterparts disjointly from their performance [18], [65]. To fill this gap, this article extends the accuracy-parameter trade-off analysis to quantized networks and proposes a metric to assess the memory allocation efficiency of BNNs for VPR applications.

III. Binary Neural Networks for VPR

This section presents a new class of BNNs for VPR and provides implementation details. To achieve memory and computational efficiency while maintaining reasonable VPR performance, we propose a multistep approach to turn a standard CNN into a compact yet effective feature extractor. Fig. 2 summarizes the process. Binarization reduces the model size and speeds up convolutions by enabling bitwise operations. Depth reduction decreases the number of layers for further model size reduction and faster computation. The subsequent performance loss due to binarization and layer removal is countered chiefly by training

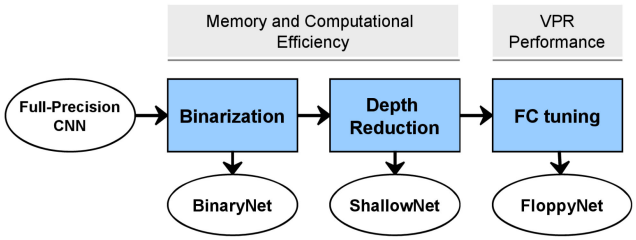

Fig. 2. Proposed approach consists of three steps: Binarization reduces the model size by about 97%. Depth reduction decreases the number of layers for further model size and MAC number reduction. The subsequent performance loss due to binarization is mainly countered by training the network with an appropriately sized fully connected stage consisting of full-precision neurons.

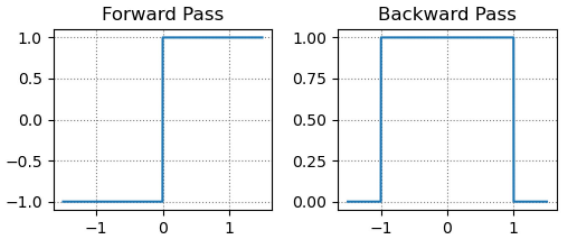

Fig. 3. Sign quantizer in forward and backward passes.

the network with an appropriately sized fully connected (FC) stage consisting of full-precision neurons.

A. Binarization

Binarization improves both memory usage and computational speed. Storing a 32-bit weight requires four bytes, while a single bit is needed for a binary one. Hence, by concatenating 32 binary weights into a floating-point variable, the resulting model size is about 97% smaller than its full-precision counterpart. Similarly, bitwise operations between weights and activations are computed in parallel with 32 binary operands resulting in a significant convolutions speedup.

Training a binary model with a reasonable performance gap from its full-precision counterpart requires applying specific techniques and some network structure adjustments. This section has the two-fold purpose of describing the implementation criteria we have taken and giving a gentle introduction to BNNs.

1) Training and Binary Function: Training BNNs with backpropagation is not applicable as it requires a sufficient precision to allow gradient accumulation to work [41]. Courbariaux and Bengio [26] solved this problem with STE [17]. The fundamental idea of STE is that the quantization function is applied in the forward pass but skipped during backpropagation. STE keeps a set of full-precision weights denoted as proxies (W_F) which are binarized (W_B) on forward pass to make a prediction and compute a loss. Any function can be used as binarization function. Courbariaux and Bengio used $sign$ function

$$W_B = sign(W_F)\,. \tag{1}$$

In the backpropagation phase, W_F are updated accordingly to the loss gradient as in a regular network

$$\frac{\partial L_{\text{oss}}}{\partial W_F} = \frac{\partial L_{\text{oss}}}{\partial W_B}\,. \tag{2}$$

Activations are binarized in the forward pass similarly to the weights. Fig. 3 shows the plots for the binarization function. In the forward pass, it behaves as the $sign$ function performing binarization. In the backward pass, the function returns a clipped identity of the gradient. Courbariaux and Bengio [26] observed that canceling the gradient when activation (a_F) exceeds 1.0 improves a model's accuracy

$$\frac{\partial L_{oss}}{\partial a_F} = \begin{cases} \frac{\partial L_{oss}}{\partial W_B}, & \text{if} |a_F| \leq 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The binary Models Presented in this article use $sign$ as a quantizer and are trained with Larq [36]. Larq is a framework built on top of Keras [25], which offers full support to train BNNs with STE.

2) Encoding Values: Binary encoding of weights and activations reduces dot products to a series of bitwise operations. In particular, representing logical "0" and "1" with -1 and 1 renders convolutions and matrix multiplications a series of XNOR and pop-count operations [26]. However, to exploit the efficiency of binary operations, a dedicated compute engine or specific hardware is required [26], [41]. A conventional compute engine stores binary weights into 32-bit variables. As a result, multiply-accumulate operations (MAC) in BNNs require the same time and resources as in a full-precision network. A proper compute engine concatenates 32 binary variables into a 32-bit register and evaluates them altogether using bitwise operations. Typically, a binary MAC is implemented as follows:

$$a_1 \mathrel{+}= \text{popcount}(\text{xnor}(a_o^{32}, w_1^{32})) \tag{4}$$

where a_o^{32} and w_1^{32} are sets of 32 inputs and weights. Although weights concatenation enables the computation of multiple binary MACs in parallel, $32\times$ speedup is unrealistic. This limitation depends on several factors, including instruction scheduling, CPU pipeline stalls, and the hardware instruction set above anything else. General purpose CPUs and GPUs have specialized instructions for fusing a floating-point MAC in a single clock cycle. Conversely, in the binary case, no such instructions exist. Hence, a binary MAC results from multiple instructions on many hardware platforms such as Nvidia GPUs [3] and ARM processors [1]. Therefore, if we let c_b represent the number of clock cycles to compute a binary MAC for a given hardware platform, then the obtainable speedup is capped at $32/c_b$.

3) Batch Normalization: Batch normalization (BatchNorm) [42] uses mini-batch statistics during training to adjust and scale activations. The central role of BatchNorm in full-precision networks is to speed up the training. In BNN, BatchNorm is essential as it improves the performance and helps training convergence [12], [67], [68]. It is worth mentioning that the parameters of BatchNorm layers cannot be binarized; however, they are few compared with the number of weights and do not contribute significantly to a model's size (see Table II).

TABLE I
TEST DATASETS AND GROUND-TRUTH TOLERANCE

Dataset	Viewpoint	Condition	Reference Images	Ground Truth
GardenPoints	Lateral	Day-Night	200	2 frames
Nordland	None	Summer Winter	309	5 frames
Old City	6-DOF	None	6708	by authors
RobotCar Cross-Seasons	Lateral	Dynamic Obj Illumination	206	5 frames

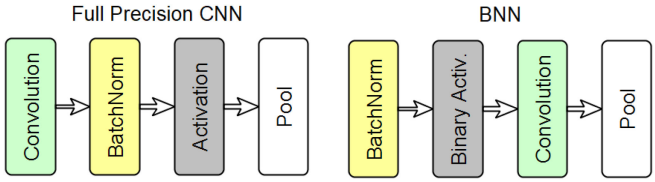

Fig. 4. Convolutional blocks in a CNN (left) and in a BNN (right).

4) Layers Order: A CB in a CNN consists of convolution, BatchNorm, activation, and pool. BNNs achieve better performance if the order of the layers is as follows: BatchNorm, binary activation, convolution, and pool [65]. This layer arrangement has a twofold purpose. First, it allows for pooling from real values before binarization. Otherwise, the result would be a tensor dense in "ones," which is proven to negatively affect the accuracy of a BNN [12]. Second, BatchNorm can replace bias as it works as a threshold for the subsequent layer [67], [68]. As bias parameters cannot be binarized, not using them reduces the memory and the number of full-precision MACs in binary networks. The BNNs implemented for this article do not use bias but use BatchNorm modules instead.

In the rest of this article, the term CB implies the presence of BatchNorm in the proper position, as shown in Fig. 4.

5) First Layer Input: Full-precision inputs are recommended to improve a binary model's accuracy [41]. The model size is unaffected since the weights are binary and the impact of computational speed is acceptable when the convolution filters is few compared to deeper layers. Accordingly with this consideration, the binary networks presented in this article have the first convolutional layer directly connected to the input image with no binary activation and BatchNorm placed in the middle.

6) Padding: In full-precision networks, convolutions are often padded with zeros. This standard practice cannot be applied to BNNs that require padded values within the encoding set to enable bitwise operations. Zero-padding, in a BNN using $\{-1, 1\}$, adds a third value along with -1 and 1 that renders convolutions incompatible with bitwise operations. Our models, therefore, use one-padding accordingly with the weights encoding $\{-1, 1\}$.

B. Depth Reduction

The primary motivation for depth reduction is to decrease the number of a model's parameters. Networks for classification are deep and can have dozens of convolutional levels [47]. However, VPR is a different task and we empirically found that it is possible achieving good performance with fewer layers and weights. Not only the model size but also the computational efficiency of the network benefits from depth reduction. For example, our best model is obtained by removing the two intermediate convolutional layers from an AlexNet-like CNN, as shown in Fig. 5(a). This operation decreases by 66% the weights amount yielding significant model size and MACs reduction (see Table III). In our experiments, the network resulting from depth reduction is denoted by ShallowNet (see Fig. 2).

C. Fully Connected Stage Tuning

BNNs are highly optimized for classification. The FC stage of classifiers is often populated with a large number of neurons. AlexNet and VGG16, for example, include 4096 units in each layer. When it comes to training a model for VPR, the hyperparameters of the FC layers should be revised. Our best binary model is trained using 256 32-bit neurons per FC layer [see Fig. 5(b)]. The optimal FC size of 256 neurons has been determined empirically by training and testing several models. The use of a full-precision instead of binary FC stage is based on the following considerations. Binary weights are a source of gradient noise [41] that renders the training more complex and longer to complete [75]. Using 32-bit FC reduces the number of binary weights that need to be learned, making training more stable. Smoother training has a lower chance to overshoot a loss function's minimum resulting in a better optimized model. Fig. 6 shows the VPR performance of the proposed model for various FC sizes. The performance peak corresponds to 256 full-precision neurons. It is relevant mentioning that FC-stage tuning is applicable only when VPR is carried out with convolutional features [see Fig. 5(c)]. This is the case of the proposed FloppyNet, which uses *pool5* features for VPR, as detailed in Section IV-E.

IV. EXPERIMENTAL SETUP

This section provides details about the experimental setup (including evaluation criteria, training, and test datasets) used for assessing the VPR performance of the BNNs presented in Section IV-D.

A. VPR Performance

VPR is cast as a loop-closure detection task [14]. Reference images showing already visited locations are searched to find the best match with the robot camera's current view, namely the query image. VPR is considered successful when a query image is paired with one of the correct reference images. The image descriptors used to match images are obtained by L2-normalization of a network's layer output

$$D = \frac{\hat{X}_l}{||\hat{X}_l||_2} \tag{5}$$

where $\hat{X}_l$ is the output of the lth layer.

Descriptors are compared using Euclidean distance; the shorter the distance, the higher the similarity between two

TABLE II
BASELINE STRUCTURE AND MODEL SIZE BEFORE AND AFTER BINARIZATION. SIZE RATIO IS BETWEEN BINARIZED AND FULL-PRECISION MODELS

	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	fc6	fc7
Layer Setup	C(11,4,96)	P(2,2)	C(5,1,256)	P(2,2)	C(3,1,384)	C(3,1,384)	C(3,1,256)	P(2,2)	FC(4096)	FC(4096)
Features Size	290400	290400	186624	43264	64896	64896	43264	9216	4096	4096
Parameters (M)	0.03	0.04	0.65	0.65	1.54	2.86	3.75	3.75	41.5	58.29
Model Size (KiB)	136.5	137.3	2538	2540	5998	11186	14646	14648	162120	227704
Total MACs (M)	105.42	105.42	553.31	553.31	702.83	927.11	1076.63	1076.63	1114.38	1131.16
Binarizable Par. (M)	0.03	0.03	0.65	0.65	1.53	2.86	3.75	3.75	41.49	58.27
Non-Binarizable Par.	0	0	192	192	704	1472	2240	2240	2752	10944
Binary Model Size (KiB)	4.25	4.25	80	80	190	355	466	466	5076	7156
Model Size Ratio (%)	3.12	3.1	3.15	3.15	3.17	3.17	3.18	3.18	3.13	3.14

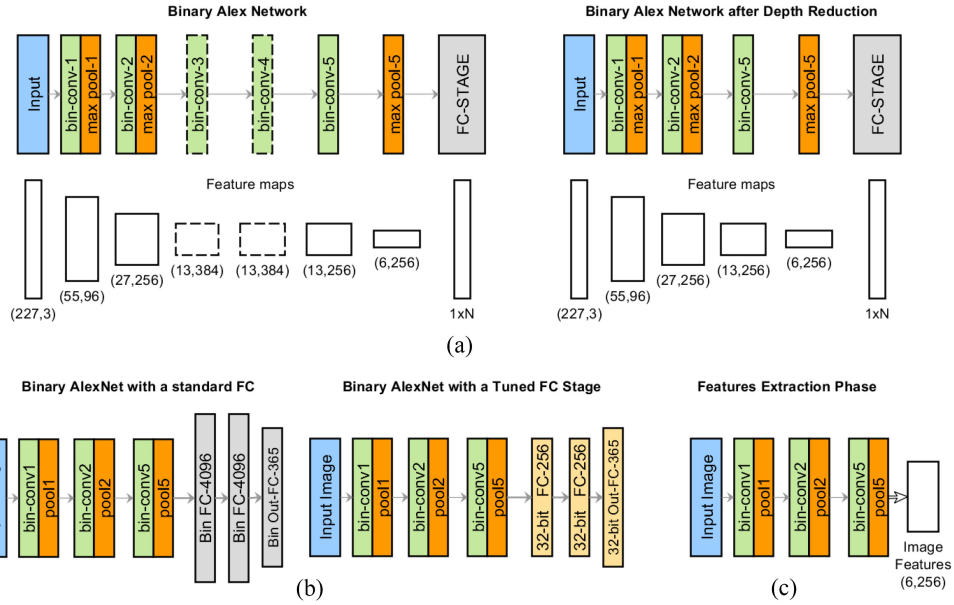

Fig. 5. (a) Depth reduction and (b) FC tuning applied to AlexNet. Depth reduction consists of removing *conv3* and *conv4* layers. (c) Three pooling layers are kept to maintain the exact shape of the *pool5* output feature map.

TABLE III
PROPOSED BNNS STRUCTURE AND MODEL SIZE

	conv1	pool1	conv2	pool2	conv5	pool5
Layer Setup	C(11,4,96)	P(2,2)	C(5,1,256)	P(2,2)	C(3,1,256)	P(2,2)
1-bit parameters (M)	0.03	0.03	0.65	0.65	1.24	1.24
32-bits parameters (M)	0	0	192	192	704	704
Model Size (KiB)	4.25	4.25	80	80	154	154
Total MACs (M)	105.42	105.42	553.31	553.31	652.99	652.99
Parameter Rate %	100	100	100	100	33.1	33.1
Size Rate % (BinaryNet)	100	100	100	100	33.05	33.05
Size Rate % (Baseline)	3.12	3.1	3.15	3.15	1.05	1.05
MAC Rate %	100	100	100	100	60.7	60.7

images

$$d = ||D_1 - D_2||_2 \tag{6}$$

where D_1 and D_2 are the image descriptors to be compared. The reference image with the shortest distance from the query is regarded as the current location.

Following the approach proposed in [33], VPR is evaluated on a whole dataset with S_{P100} index. It represents the ratio of places that are correctly recognized against the ground-truth.

B. Memory Allocation Efficiency

VPR performance is also evaluated in relation to memory requirements. We define memory efficiency as the ratio of the model size to S_{P100}

$$\eta_m = \frac{M_{\text{size}}}{S_{P100}}. \tag{7}$$

η_m measures the memory cost per S_{P100} point, expressing the trade-off between memory usage and VPR performance. The

TABLE IV
S_{p100} FOR EVERY LAYER IN BASELINE

	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	fc6	fc7
GardenPoints	14.5	29	56.5	62.5	67	79.5	66	84	73	56.5
RobotCar	80.1	89.3	87.4	90.3	91.3	92.2	82	93.2	90.8	74.3
Nordland	69	75.5	86.5	91	91	95	54	85	40	29
Old City	7	11	7	9	11	18	13	33	39	37
Average	42.7	51.2	59.4	63.2	65.1	71.2	53.8	**73.8**	60.7	49.2

The bold entity indicates the highest value in the row.

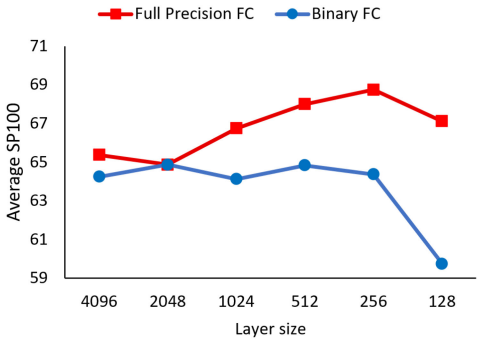

Fig. 6. Average S_{P100} across all the test datasets for full-precision and binary fully connected stages at different layer sizes.

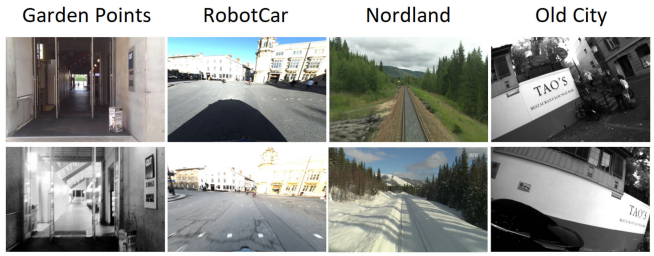

Fig. 7. Corresponding image pair from each test dataset.

lower is η_m, and the more efficient is the model at using memory. Memory efficiency is a generalization of the trade-off analysis between accuracy and parameters density [19] to low-precision networks whose memory footprint also depends on weight quantization, and hence, the use of model size instead of the number of weights in (7). Moreover, η_m can be applied to networks having the same structure but different weight precision to determine the relationship between VPR performance and quantization, providing additional information to characterize a low-precision network or choose the optimal quantization for an application.

C. Inference Latency and Power Usage

Inference latency, T_i, and power usage (P_w) measurements are taken from deployed models. T_i is the time required by a model to complete a forward pass. The time intervals to load an image and consume the output are excluded from the measurement so that T_i reflects the actual computational effort for an image representation. P_w is measured directly on the hardware platform and used to determine the inference energy cost

$$E_i = P_w T_i \ . \tag{8}$$

1) Training Data: The dataset used to train all the models is Places365 [5], [84]. It is a place-themed dataset consisting of 1 803 460 images divided into 365 categories with between 3068 and 5000 images in each category. The validation set includes 100 images per location class.

2) Test Data: On long-term runs, a robot visits a place at different times or from different directions. These factors yield changes in the appearance of places captured by the robot's camera. In order to provide comprehensive results, test data include four datasets, each containing environmental and/or viewpoint changes. All datasets have two subsets that correspond to different traverses of the environment (see Fig. 7). One is the reference dataset representing the previous knowledge of the environment while the other is the query dataset that represents the current traverse. The datasets include a different number of query images. To compute fair average performance indicators (e.g., Table IV), we randomly sampled 200 query images from each of them for a total of 800 images.

The datasets are detailed below and summarized in Table I.

1) GardenPoints Walking [73]: This dataset includes three traverses of the Queensland University of Technology (QUT). The experiments employed Right-Day and Right-Night to test VPR under illumination changes and mild lateral shifts. Ground-truth is built by frame correspondences with a tolerance of ± 2 frames [53].

2) Nordland [71]: A set of four traverses captured along a rail track in Norway in every season. The experiments employed Summer and Winter journeys as reference and query datasets, respectively. The ground-truth is built with a tolerance of ± 5 frames [53].

3) Old City [56]: A urban dataset with two traverses showing the same location from different perspectives to generate the viewpoint variations experienced by a 6-DOF (degrees-of-freedom) aerial robot. The ground-truth data are available from the authors [10].

4) RobotCar Cross-Seasons [49]: A subset of the Oxford RobotCar dataset [54] consisting of two sequences of 206 sunny query images and 202 dusk reference images recorded on board of a car driving in an urban environment. This dataset includes illumination changes, mild lateral viewpoint shifts, and dynamic elements such as pedestrians, cars, and shadows. Ground-truth is built by frame correspondences with a tolerance of ± 5 frames.

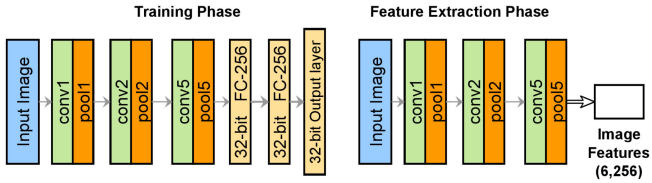

Fig. 8. FloppyNet layer structure. The output features used for VPR are from the *pool5* layer.

5) Binary Network and Comparison Baseline: The networks used for the experiments are based on the AlexNet archetype [48], which is one of the most used network types for VPR [13], [24], [45], [74]. AlexNet-type networks' structure consists of several CBs alternated with pool layers followed by a FC stage with one or more hidden layers.

The baseline CNN, denoted by Baseline in this article, is very similar to a standard AlexNet [48] except for the use of BatchNorm and pool layers with a 2×2 nonoverlapping kernel for higher accuracy [48]. Baseline network has five CBs followed by a FC stage with two hidden layers, including 4096 neurons each. The detailed structure is shown in Table II using the following notation. $C(k, s, h)$ indicates a CB with kernel size k, stride s and h channels (filters). A similar notation is used for max pooling: $P(k, s)$. FC layers are indicated with $FC(n)$, where n is number of neurons. The model sizes and MACs reported in Table II are cumulative per network layer. For example, if the baseline is cut to use *fc6* features, the corresponding size of the model is 158.32 MiB, and the MACs are 1.1 billion. BinaryNet is the binary version of Baseline. The bottom part of Table II shows the number of binarizable parameters. The remaining 32-bit parameters are due to BatchNorm as described in Section III-A3. However, their contribution to the binary model size is negligible. BinaryNet sizes vary from the 3.12% of Baseline at $conv1$, which is not preceded by a BatchNorm layer, to 3.18% at $pool5$.

6) FloppyNet: FloppyNet consists of three binary CBs and three pool layers, as shown in Fig. 8. Binarization, jointly with depth reduction applied to Baseline, resulted in a compact model of 154 KiB. The layers removed in the depth reduction step are *conv3* and *conv4*. The output layer of FloppyNet is denoted as *pool5* by convention. We kept the same name as in Baseline network since they have the same structure and provide feature vectors with the same shape and element number: $6 \times 6 \times 256 = 9216$ elements.

The primary motivation for FloppyNet is to reduce the model size further and shorten the inference latency. With two fewer convolutional layers, FloppyNet uses 33% of the memory of BinaryNet at the *pool5* layer, computing 39% fewer MACs (see Table III).

Binarization and depth reduction cause a performance loss that is mostly compensated by tuning the FC stage properly for the training. Our best model is obtained with 256 full-precision neurons in both *fc6* and *fc7*. This training approach's effectiveness is demonstrated in Section VI where FloppyNet is compared against ShallowNet, which is trained without tuning the FC stage.

V. BNN Layers Analysis

The first question to answer when a convolutional network is employed as a feature extractor is: which layer is the most suitable to build a distinctive image descriptor? This section provides a VPR performance assessment of the features from every layer in both Baseline network (Section IV-D) and its binary counterpart, BinaryNet. The results obtained drove the design of FloppyNet toward the use of *pool5* as an output layer.

CNNs can learn features at different levels of abstraction. Convolutional features retain some spatial information. However, as the depth increases, pool layers induce the loss of such a spatial information in favor of translation invariance. In FC layers, the activation of a neuron depends on every neuron in the previous level. Hence, the spatial information vanishes while improving the invariance to viewpoint changes and translation in particular [39]. The second question we need to answer is: how does binarization impact layers' features and VPR matching performance?

The answers to these questions are given in Tables IV and V, which show S_{P100} for every layer of Baseline and BinaryNet, respectively. In Baseline, FC layers and deeper convolutions handle viewpoint changes better than initial layers. *Fc6* and *fc7* obtain the highest performance under the extreme viewpoint changes that characterize Old City. *Pool5* is the best on GardenPoints, which includes mild viewpoint shifts other than day-light variations. On the other hand, shallower layers deal better with appearance changes. Nordland includes only seasonal variations, and the best layer is *conv4* with $S_{P100} = 95\%$. These results partially confirm the findings of a previous study on a standard AlexNet [39], which indicates *conv3* as the best layer to deal with appearance changes while *pool5* and, in some cases, *fc6* as the best choice to deal with viewpoint changes.

Binarization negatively affects VPR performance, but the characteristics of the layers are more or less unchanged. As shown in Table V, *pool5* achieves the highest performance on the same dataset as for Baseline. Similarly, FC layers outperform the others on Old city.

Overall, *pool5* is the layer that guarantees the highest average performance across the four datasets. The average S_{P100} is 73.8% for Baseline and 64.3% for its binary counterpart. The gap is moderate (9.5%), especially considering that BinaryNet at *pool5* requires only 3.18% of memory used by the Baseline (see Table II). The average S_{P100} is computed across all the datasets as they formed a single environment to simulate a workspace, including a wide variety of viewpoint and appearance changes. For a robot navigating such a workspace, $pool5$ features guarantee the most reliable and consistent VPR performance. Accordingly, we designed FloppyNet with the same shape progression of the feature maps as in BinaryNet to have the output layer with similar characteristics as *pool5*. As detailed in Section IV-E, this is obtained by removing two inner CBs: *conv3* and *conv4*.

VI. VPR Performance Comparison

FloppyNet is compared against several other networks. These include HybridNet, VGG-16, CALC, Baseline (Section IV-D), BinaryNet, ShallowNet, and a 8-bit implementation of

TABLE V
S_{p100} FOR EVERY LAYER IN BINARYNET

	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	fc6	fc7
GardenPoints	3	11.5	43	51.5	75.5	74.5	76	79.5	66.5	39.5
RobotCar	61.2	73.3	81.4	81.9	82.4	82.4	83	83	84	63.6
Nordland	53	60	71.5	71	77.5	77	72	71.5	38.5	14
Old City	9	10.5	11.5	15	16	15	20.5	23	49	44.5
Average	31.6	38.8	51.9	54.9	62.9	62.2	62.9	**64.3**	59.5	40.4

The bold entity indicates the highest value in the row.

TABLE VI
MODELS' PERFORMANCE AND EFFICIENCY FOR RASPBERRY PI 4 IMPLEMENTATIONS

	bits	av. SP100	Params [M]	Size [KiB]	η_m [KiB]	MACs	T_i [ms]	FPS	P_w [W]	E_i [mJ]
HybridNet	32	75.4	5.07	16957	224.89	1098 M	137.8	7.26	2.616	360.5
Baseline	32	73.8	3.75	14648	198.48	1077 M	132.4	7.55	2.616	346.4
VGG-16	32	77.9	14.7	57487	737.96	15.3 B	2211	0.45	2.616	5784
BinaryNet	1	64.3	3.75	466	7.25	1077 M	21.6	46.3	2.536	54.8
ShallowNet	1	62.9	1.24	154	2.45	653 M	18.2	54.95	2.536	46.2
FloppyNet	1	68.7	1.24	154	2.24	653 M	18.2	54.95	2.536	46.2
FloppyNet-8	8	71.4	1.24	1213	16.99	653 M	33.2	30.12	2.876	95.5
FloppyNet-32	32	72.9	1.24	4843	66.43	653 M	76.1	13.14	2.616	199.1
CALC	32	40.5	0.137	537	13.26	186 M	45.1	22.17	2.616	118.0

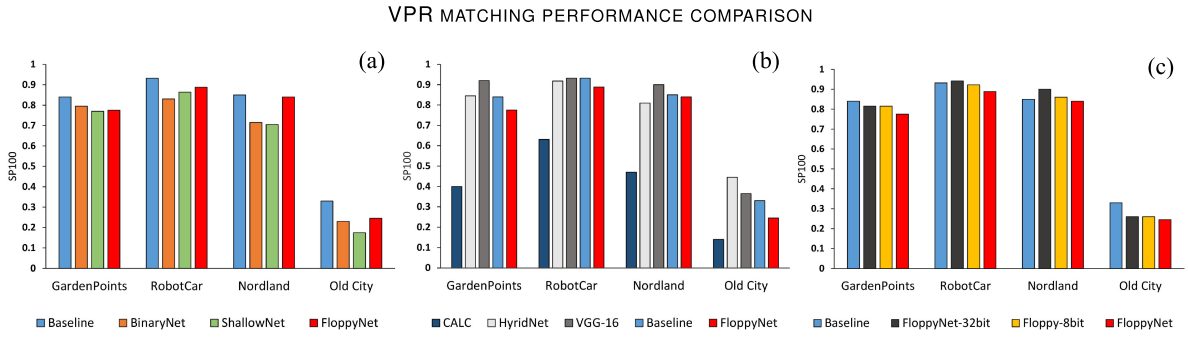

Fig. 9. FloppyNet is compared on a variety of imaging conditions against the (a) Baseline, BinaryNet, and ShallowNet, (b) full-precision networks, and (c) several FloppyNet versions with different quantization levels.

FloppyNet. ShallowNet is the version of FloppyNet trained with regular FC layers of 4096 binary neurons as described in Section III-B. HybridNet [23] is a version of AlexNet with an additional CB trained on ImageNet and tuned on SPED dataset [23]. To avoid training data influencing the results, we trained an HybridNet model by replacing ImageNet with Places365, which is the dataset used to train the other models considered for the experiments. The results for HybridNet are obtained with *pool6* features. VGG-16 [69] is a very deep network if compared to FloppyNet since it includes 13 CBs. It is relevant to include VGG-16 in the comparison because several multistaged VPR methods widely use it as a feature extractor. Some examples are R-MAC [74], Cross-Region-Bow [24], and NetVLAD [13]. The VGG-16 model has been trained from scratch using Places365, and the features used for the tests are from the very last pool layer. CALC [58] is a lightweight CNN designed to address VPR with low resource requirements. CALC includes about 137 K parameters: a small fraction of 3.75 M of Baseline and 1.24 M of FloppyNet (see Table VI). We used the model trained on Place365 shared by the authors. The 8-bit version of FloppyNet is included in the comparison to show that BNNs also scale well to 8-bits quantization, demonstrating potential applicability of binarization for VPR as a more efficient yet effective approach.

A. Comparison With the Baseline

FloppyNet aims to achieve similar performance as the starting Baseline network (Section IV-D) with higher efficiency. Fig. 9(a) shows comparative results between FloppyNet, Baseline, and the intermediate design steps: BinaryNet and ShallowNet (Section IV-E). Binarization and depth reduction negatively impact the VPR performance. BinaryNet and ShallowNet score the lowest S_{P100} on every dataset, exhibiting a substantial gap from Baseline. FloppyNet outperforms BinaryNet and ShallowNet on every dataset confirming that tuning the FC stage during the training (Section III-C) mitigates the performance loss due to binarization and depth reduction.

In general, Baseline has better performance than FloppyNet. However, on GardenPoints, Nordland, and Robotocar, the difference is small and is due to a few places with some particular characteristics.

The most difficult locations to recognize for FloppyNet are those presenting strong viewpoint variations. On Old City, the S_{P100} difference is 8.5%, which is the highest among the test datasets. An analysis on GardenPoints mismatches confirms such a FloppyNet's weakness. GardenPoints presents mild later shifts except in a few locations where FloppyNet fails while Baseline succeeds (see Fig. 10(a)).

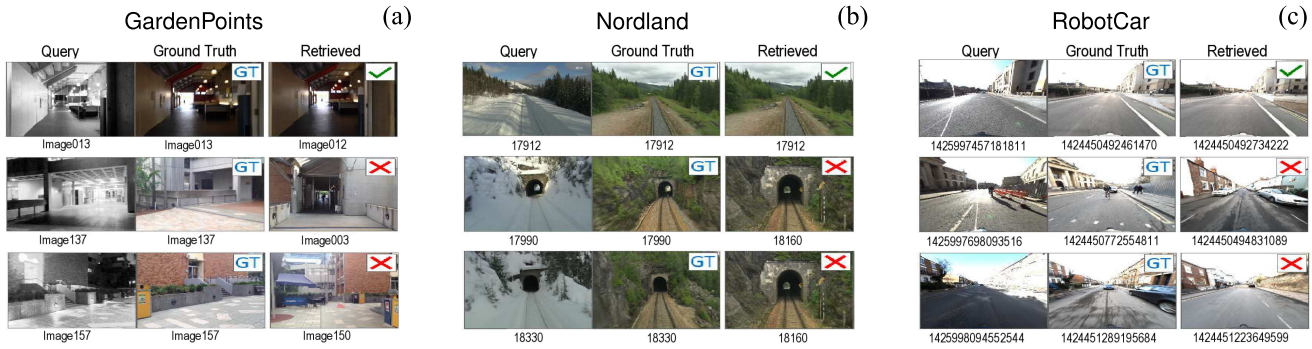

Fig. 10. Some significant query results from (a) GardenPoints, (b) Nordland, and (c) RobotCar Cross-Seasons datasets.

FloppyNet scores almost the same S_{P100} as Baseline on Nordland, 84% versus 85%. The mismatches are mainly between locations showing tunnel entrances. Fig. 10(b) shows two examples of mismatch. FloppyNet retrieves the same reference image on two different queries, including a tunnel.

FloppyNet scores a good S_{P100} on RobotCar as well. The S_{P100} difference with Baseline is caused by a series of wrong matches in the two locations shown in Fig. 10(c). The high illumination contrast and the occlusions due to dynamic elements, such as cars and shadows, are possibly the cause of FloppyNet's failures.

B. Comparison With Full-Precision CNNs

Full-precision and deeper networks obtain better VPR performance than the proposed binary network [see Fig. 9(b)]. Substantial gaps are exhibited on Garden Points and Old City by VGG-16 and HybridNet, respectively. On the other hand, those two networks are far larger than FloppyNet even without considering the weights' precision. FloppyNet scores an average S_{P100} of 68.7% using 1.24 M binary parameters and computing 653 M MACs. The highest average S_{P100} is achieved by VGG-16, 77.9%. However, VGG-16 includes 14.7 M weights and computes 15.3 B floating-point MACs resulting in two orders of magnitude longer inference latency (see Table VI). The small number of parameters penalizes CALC, which our network outperforms by a large margin on every dataset. Moreover, its low VPR performance is not compensated by a sufficient computational efficiency as CALC is about two times slower than FloppyNet, as detailed in Section VII-A.

C. Weight Precision Impact

Fig. 9(c) presents a comparison between our best FloppyNet model and its 8-bit and full-precision versions to examine the impact of weight quantization on place recognition capabilities. Increasing the quantization bits has a positive effect on VPR performance. FloppyNet-8 bit outperforms FloppyNet in every test scenario and nearly closes the gap with Baseline (Section IV-D) except on Old-City. Indeed, the reduced depth affects VPR performance on significant viewpoint changes, as discussed in Section VI-A. Overall, FloppyNet has slightly lower performance than the higher precision models except on GardenPoints, where the gap from the 8-bit implementation is the largest, 6%. On the other hand, 8-bit quantization yields eight times larger models and almost doubles the inference latency and energy consumption, as shown in Section VII. The higher resource demand of 8-bit quantization might be unsuitable for extremely cheap hardware, where binarization can be used instead. Finally, further increasing the precision of the weights does not significantly improve the VPR capabilities of the proposed network. As shown in Fig. 9(c), 8-bit and 32-bit models score similar S_{P100} on every dataset.

D. Memory Efficiency

Table VI presents the memory efficiency for all the compared networks. S_{P100} is the average score on the four datasets. Binary networks have extremely low η_m values compared to any full-precision network. FloppyNet requires 2.24 KiB per S_{P100} point, while CALC, the most memory-efficient CNN, requires 13.26 KiB. ShallowNet has the same size as FloppyNet but lower S_{P100}, hence it uses memory less efficiently: $\eta_m = 2.45$ KiB. FloppyNet-8 bit performs better on average than the 1-bit model by 2.7% but requires about eight times the memory. The considerably higher $\eta_m = 16.99$ KiB reflects this trend, indicating that the increase of the model size does not correspond to a proportional increase in VPR performance.

E. Binarization, Depth Reduction, and FC-256

Fig. 11 shows S_{P100} relatively to Baseline (Section IV-D) resulting from using binarization (Bin), depth reduction (Depth), and FC-stage tuning (FC256) separately and their relevant combinations. The features used to obtain the results are from the *pool5* layer.

Depth reduction (Depth) yields a full-precision network with better performance on Nordland and RobotCar. Depth reduction makes the output layer of a model retaining more spatial information compared to Baseline. Shallower layers are more suitable to deal with appearance changes (see Table IV). For this reason, depth reduction improves the performance on Nordland and Robotcar datasets that present significant appearance changes while none or mild viewpoint variations. Training a full-precision model with 256 neurons in the FC stage (FC256) helps VPR significantly to tackle extreme 6-DOF viewpoint variations. FC256 model achieves 25% higher S_{P100} on Old City than the original model trained with 4096 neurons in the

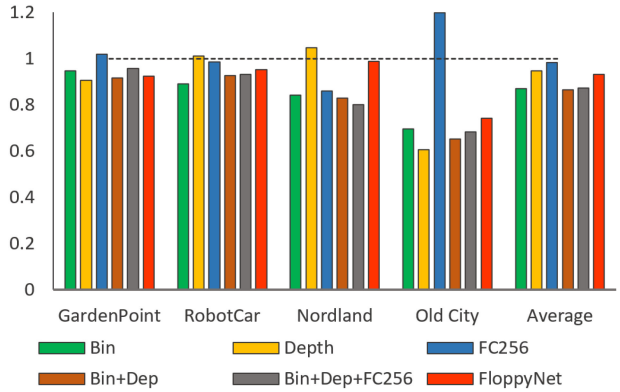

Fig. 11. S_{P100} is relative to Baseline (dotted line) of several combinations of the three techniques used by FloppyNet: depth reduction, binarization, and training using a fully connected stage with 256 neurons. The features used for VPR are from *pool5* layer.

FC stage. In classifiers, the FC stage is usually sized as large as possible to maximize accuracy while avoiding overfitting. Conversely, the empirical evidence shows that VPR benefits from a smaller FC stage. These results suggest that the FC stage has different roles in classification and VPR tasks.

Depth reduction does not help S_{p100} scores of the binarized models. The values of S_{p100} for binary (bin) and shallow binary networks (bin+dep) are very close to each other on every test dataset (Nordland in particular), which is rather unexpected considering the full-precision case (Depth). The red bars in Fig. 11 represent FloppyNet, which implements all the steps of the proposed approach. The addition of FC-stage tuning counters the S_{p100} loss due to binarization and depth reduction in every tested scenario (except Garden Point), supporting the effectiveness of the proposed training approach.

VII. Benchmark Analysis

The software framework used to deploy binary models is Larq Compute Engine (LCE) [15]. LCE consists of a model compiler and a kernel to compute binary convolutions within the Tensorflow Lite runtime environment (TFlite) [9], [30]. LCE was a natural choice as it is part of the Larq ecosystem [36] we used to train our models. The 8-bit and full-precision implementations use the built-in TFlite compute kernel.

The evaluation criteria are the inference latency and power usage. Inference latency indicates the time required to complete an inference, namely to compute an image representation. Power usage is measured on a model execution and is used to determine the energy cost per inference [see (8)]. The platform employed for the experiments is a Raspberry PI4 (RPI4) that sits on an ARMv8 Cortex-A72 running at 1.5 GHz [8]. The operating system is Ubuntu 20.04 Linux, 64-bit.

A. Inference Latency and Computation Speedup

Fig. 12(a) compares the inference latency of Baseline (Section IV-D) and two FloppyNet implementations: 1-bit and 8-bit. The values reported in the figure are an average of 100 runs using four threads, namely employing all the cores available in the RPI4's CPU. FloppyNet has a latency of 18.2 ms, which is about seven times shorter than Baseline's (132.4 ms).

Table VI presents the inference latency for all the tested models. FloppyNet is considerably faster than any other full-precision CNN including CALC, which computes only 186 M MACs, 28.5% of FloppyNet's MACs. However, it takes a considerably longer inference time than our network, 45.1 ms against 18.2 ms, while achieving considerably lower VPR performance [see Fig. 9(b)]. BinaryNet's latency enables an analysis of the speedup contributions provided by binarization and depth reduction. BinaryNet has the same structure as Baseline except for the binary weights and runs in 21.6 ms, resulting in six times faster than its full-precision counterpart. Depth reduction removes two convolutions from BinaryNet, shortening the execution by a further 16%. Latency measures demonstrate that FloppyNet also scales well to 8-bit quantization. FloppyNet-8 bit completes an inference in 33.2 ms, resulting almost twice slower than FloppyNet.

B. Power Usage

The power usage is measured with an ampere meter connected to the USB-C power port of the RPI4. The energy spent per inference by a model, E_i, depends on the power absorbed by the RPI4, P_w, and the inference latency, T_i [see (8)]. P_w is stable during an inference depending only on a model's weights precision because of the computational kernel used: LCE for binary models and TFlite for 8-bit and full-precision models. The measured P_w is 2.54 W, 2.62 W, and 2.88 W for binary, 32-bit, and 8-bit models, respectively. As T_i varies in a wider range than P_w (see Fig. 12), E_i depends mainly on a model's inference latency. The rightmost columns of Table VI show the power absorption and inference energy for every model. BNNs consume less energy than any other considered network. In particular, FloppyNet uses 46.2 mJ per inference, which is considerably more energy-efficient than the 8-bit implementation and Baseline spend of 95.5 and 346.4 mJ, respectively.

VIII. Comparison With Handcrafted Descriptors

This section compares FloppyNet with several handcrafted image descriptors relevant for VPR applications [81]. They include HOG [29], GIST [61], and CoHOG [79]. The results show that our model has significantly better VPR capabilities while having comparable or higher computational efficiency than the considered handcrafted descriptors.

A. Handcrafted Descriptors Setup

The platform employed for measuring the processing time is a Raspberry PI4 (RPI4), as in the previous section. The handcrafted descriptors setting used for the experiments are as follows.

1) HOG: We used the OpenCV 4.5.0 implementation with a cell size of 16×16 and block size of 32×32, as suggested in [81]. The input image size is set to 256×256 pixels, which is similar to FloppyNet's input size of 227×227 pixels.

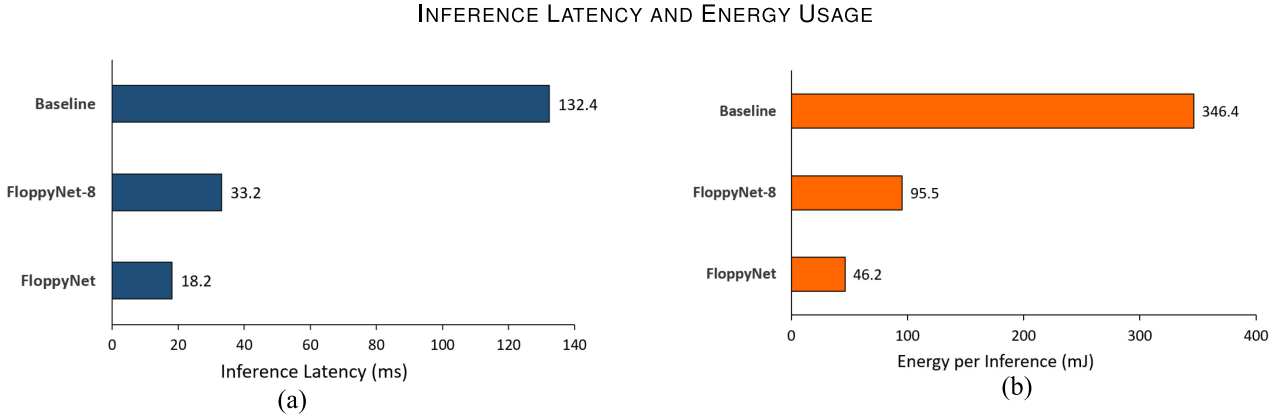

Fig. 12. (a) Inference latency and (b) energy usage of FloppyNet models and Baseline executed with four threads on a Raspberry PI4.

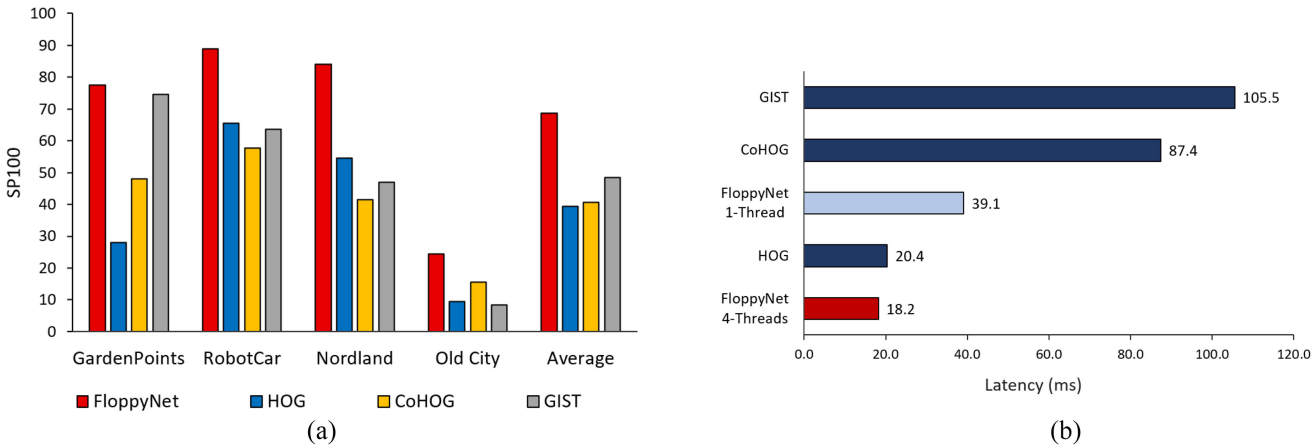

Fig. 13. (a) VPR performance and (b) computational latency of HOG, CoHOG, and GIST compared with FloppyNet. The latency is measured on a Rasperry PI4.

2) CoHOG: We used the code shared by the authors with their recommended settings [2]: cell size of 16×16, 8 bins, and entropy threshold of 0.4. The image size is 256×256 pixels.

3) Gist: It is available in the *C* library *Lear's GIST* through the *pyleargist* python wrapper [6]. Gist is used with the parameters indicated by the authors [4]: four blocks and eight orientations per scale. We set the image size to 128×128 pixels to keep the Gist's latency comparable with the other methods. Indeed, using 256×256 images extends Gist's processing time by roughly four times.

B. Results Discussion

Fig. 13 shows the results. FloppyNet achieves substantially better VPR performance on every dataset scoring an average S_{P100} of 68.7%. Gist achieves the highest VPR performance among the handcrafted descriptors but results in the slowest one taking 105 ms to process an image.

Our network is the fastest technique when running using all the four RPI4's cores. However, Gist and HOG implementations cannot run on multiple threads. We reported the inference latency for FloppyNet running on a single thread in Fig. 13(b) for a fair comparison. Only HOG runs faster than our network taking 20.4 ms to process an image instead of the 39.1 ms required by FloppyNet (1-thread). On the other hand, HOG exhibits a wide VPR performance gap on every dataset. HOG scores an average S_{P100} of 39.4%, whereas FloppyNet achieves $S_{P100} = 68.7\%$. We conclude that the shorter latency of HOG does not compensate for the poor VPR performance it achieves compared to our network.

IX. Conclusion

In this article, we proposed FloppyNet, a compact binary network, to solve the VPR problem. FloppyNet achieved comparable VPR performance to deeper and full-precision networks in changing environments with drastically lower memory requirements and substantial computational speedup. Such lightweight networks open up several opportunities for embedded systems and edge computing in general. FloppyNet may be employed to enable VPR on very cheap hardware or replace standard CNNs to free up resources to allocate for additional functionalities to improve a robot's navigation system or increase the frame rate on low-cost embedded applications. For example, NetVLAD is a two-stage image descriptor that uses VGG to extract image features that are subsequently postprocessed to compute a robust image representation. VGG is a large network that requires a relatively

long time to extract image features. If a BNN, such as FloppyNet, is used instead, NetVLAD's memory requirements and computational efficiency improve dramatically. This example suggests that a natural extension of this article investigates the applicability of BNNs in multistage descriptors that use a CNN as a feature extractor.

REFERENCES

[1] "Arm Cortex-A76 software optimization guide," Accessed: Feb. 21, 2022. [Online]. Available: https://developer.arm.com/documentation/swog307215/a

[2] "CoHOG source code," Accessed: Feb. 21, 2022. [Online]. Available: https://github.com/MubarizZaffar/VPR-Bench/tree/main/VPR_Techniques/CoHOG_Python

[3] CUDA Toolkit Documentation, Accessed: Feb. 21, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions

[4] GIST MATLAB implementation, Accessed: Feb. 21, 2022. [Online]. Available: http://people.csail.mit.edu/torralba/code/spatialenvelope/

[5] *Places365 DevKit*, Accessed: Feb. 21, 2022. [Online]. Available: https://github.com/zhoubolei/places_devkit

[6] Pyleargist, Accessed: Feb. 21, 2022. [Online]. Available: https://pypi.org/project/pyleargist/

[7] PYNQ, Accessed: Feb. 21, 2022. [Online]. Available: http://www.pynq.io/

[8] *Raspberry Pi 4 Tech Specs*, Accessed: Feb. 21, 2022. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

[9] Tensorflow Lite, Accessed: Feb. 21, 2022. [Online]. Available: https://www.tensorflow.org/lite

[10] "V4RL wide-baseline place recognition dataset," Accessed: Feb. 21, 2022. [Online]. Available: https://github.com/VIS4ROB-lab/place_recognition_dataset_ral2019

[11] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/about/bib

[12] M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal, "An empirical study of binary neural networks' optimisation," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJfUCoR5KX

[13] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5297–5307.

[14] D. Bai, C. Wang, B. Zhang, X. Yi, and X. Yang, "Sequence searching with CNN features for robust and fast visual place recognition," *Comput. Graph.*, vol. 70, pp. 270–280, 2018.

[15] T. Bannink *et al.*, "Larq compute engine: Design, benchmark, and deploy state-of-the-art binarized neural networks," *J. Open Source Softw.*, vol. 5, no. 45, Jan. 2020, Art. no. 1746.

[16] H. Bay, A. Ess, T. Tuytelaars, and L. van Gool, "Speeded-up robust features (SURF)," *Comput. Vis. Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[17] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[18] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "Back to simplicity: How to train accurate BNNs from scratch?," 2019, *arXiv:1906.08637*.

[19] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[20] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2020.

[21] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*.

[22] T. Chen *et al.*, "{ TVM }: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th { USENIX } Symp. Operating Syst. Des. Implementation*, 2018, pp. 578–594.

[23] Z. Chen *et al.*, "Deep learning features at scale for visual place recognition," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 3223–3230.

[24] Z. Chen, F. Maffra, I. Sa, and M. Chli, "Only look once, mining distinctive landmarks from convnet for visual place recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 9–16.

[25] F. Chollet *et al.*, "Keras," Accessed: Feb. 21, 2022. 2015. [Online]. Available: https://github.com/fchollet/keras

[26] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to 1 or −1," 2016, *arXiv:1602.02830*.

[27] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*.

[28] M. Cummins and P. Newman, "Appearance-only SLAM at large scale with fab-map 2.0," *Int. J. Robot. Res.*, vol. 30, no. 9, pp. 1100–1123, 2011.

[29] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, vol. 1, pp. 886–893.

[30] R. David *et al.*, "Tensorflow lite micro: Embedded machine learning on tinyml systems," 2020, *arXiv:2010.08678*.

[31] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rkgO66VKDS

[32] B. Ferrarini, M. Waheed, S. Waheed, S. Ehsan, M. Milford, and K. D. McDonald-Maier, "Visual place recognition for aerial robotics: Exploring accuracy–computation trade-off for local image descriptors," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2019, pp. 103–108.

[33] B. Ferrarini, M. Waheed, S. Waheed, S. Ehsan, M. J. Milford, and K. D. McDonald-Maier, "Exploring performance bounds of visual place recognition using extended precision," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1688–1695, Apr. 2020.

[34] W. T. Freeman and M. Roth, "Orientation histograms for hand gesture recognition," in *Proc. Int. Workshop Automat. Face Gesture Recognit.*, 1995, vol. 12, pp. 296–301.

[35] J. Fromm, M. Cowan, M. Philipose, L. Ceze, and S. Patel, "Riptide: Fast end-to-end binarized neural networks," in *Proc. Mach. Learn. Syst.*, vol. 2, pp. 379–389, 2020. [Online]. Available: https://proceedings.mlsys.org/paper/2020/hash/2a79ea27c279e471f4d180b08d62b00a-Abstract.html

[36] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *J. Open Source Softw.*, vol. 5, no. 45, Jan. 2020, Art. no. 1746.

[37] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[38] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. 5th Int. Conf. Neural Inf. Process. Syst.*, 1992, pp. 164–171.

[39] Y. Hou, H. Zhang, and S. Zhou, "Convolutional neural network-based image representation for visual loop closure detection," in *Proc. IEEE Int. Conf. Inf. Automat.*, 2015, pp. 2238–2245.

[40] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[41] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[43] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Proc. 23rd IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3304–3311.

[44] A. Karki, C. Palangotu Keshava, S. Mysore Shivakumar, J. Skow, G. Madhukeshwar Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2019, pp. 137–138.

[45] A. Khaliq, S. Ehsan, Z. Chen, M. Milford, and K. McDonald-Maier, "A holistic visual place recognition approach using lightweight CNNs for significant viewpoint and appearance changes," *IEEE Trans. Robot.*, vol. 36, no. 2, pp. 561–569, Apr. 2020.

[46] A. Khaliq, S. Ehsan, M. Milford, and K. D. McDonald-Maier, "CAMAL: Context-aware multi-scale attention framework for lightweight visual place recognition," 2019, *arXiv:1909.08153*.

[47] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[49] M. M. Larsson, E. Stenborg, L. Hammarstrand, M. Pollefeys, T. Sattler, and F. Kahl, "A cross-season correspondence dataset for robust semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9524–9534.

[50] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

[51] F. Li and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*.

[52] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[53] S. Lowry and M. J. Milford, "Supervised and unsupervised linear learning techniques for visual place recognition in changing environments," *IEEE Trans. Robot.*, vol. 32, no. 3, pp. 600–613, Jun. 2016.

[54] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford RobotCar dataset," *Int. J. Robot. Res.*, vol. 36, no. 1, pp. 3–15, 2017.

[55] F. Maffra, Z. Chen, and M. Chli, "Viewpoint-tolerant place recognition combining 2D and 3D information for UAV navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 2542–2549.

[56] F. Maffra, L. Teixeira, Z. Chen, and M. Chli, "Real-time wide-baseline place recognition using depth completion," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1525–1532, Apr. 2019.

[57] C. McManus, B. Upcroft, and P. Newmann, "Scene signatures: Localised and point-less features for localisation," in *Proc. Robot.: Sci. Syst.*, Berkeley, CA, USA, 2014, pp.–1–9.

[58] N. Merrill and G. Huang, "Lightweight unsupervised deep loop closure," in *Proc. Robot.: Sci. Syst.*, Pittsburgh, PA, USA, 2018.

[59] M. J. Milford and G. F. Wyeth, "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 1643–1649.

[60] A. C. Murillo, J. J. Guerrero, and C. Sagues, "Surf features for efficient robot localization with omnidirectional images," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2007, pp. 3901–3907.

[61] A. Oliva and A. Torralba, "Building the gist of a scene: The role of global image features in recognition," *Prog. Brain Res.*, vol. 155, pp. 23–36, 2006.

[62] I. Palit, Q. Lou, R. Perricone, M. Niemier, and X. S. Hu, "A uniform modeling methodology for benchmarking dnn accelerators," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2019, pp. 1–7.

[63] A. Pappalardo, "Xilinx/brevitas." Accessed: Feb. 21, 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[64] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Proc. Syst. 32*, H. Wallach H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[65] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[66] D. Saad and E. Marom, "Training feed forward nets with binary weights via a modified CHIR algorithm," *Complex Syst.*, vol. 4, no. 5, pp. 573–586, 1990.

[67] E. Sari, M. Belbahri, and V. P. Nia, "A study on binary neural networks initialization," 2019, *arXiv:1909.09139*.

[68] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, 2019, Art. no. 661.

[69] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.

[70] G. Singh and J. Kosecka, "Visual loop closing using gist descriptors in manhattan world," in *Proc. ICRA Omnidirectional Vis. Workshop*, 2010, pp. 44–48.

[71] S. Skrede, "Nordlandsbanen: Minute by minute season by season," Accessed: Feb. 21, 2022. [Online]. Available: https://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/

[72] N. Sünderhauf and P. Protzel, "Brief-gist-closing the loop by simple means," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 1234–1241.

[73] N. Sünderhauf, S. Shirazi, F. Dayoub, B. Upcroft, and M. Milford, "On the performance of convnet features for place recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 4297–4304.

[74] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of CNN activations," in *Proc. Int. Conf. Learn. Representations*, San Juan, Puerto Rico, 2016, pp. 1–12.

[75] M. Toneva, A. Sordoni, R. T. des Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BJlxm30cKm

[76] Y. Umuroglu *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[77] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized blas," *Softw.: Pract. Experience*, vol. 35, no. 2, pp. 101–121, 2005.

[78] C. Xia, J. Zhao, H. Cui, X. Feng, and J. Xue, "DNNTune: Automatic benchmarking DNN models for mobile-cloud computing," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 1–26, 2019.

[79] M. Zaffar, S. Ehsan, M. Milford, and K. McDonald-Maier, "CoHOG: A light-weight, compute-efficient, and training-free visual place recognition technique for changing environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1835–1842, Apr. 2020.

[80] M. Zaffar, S. Ehsan, M. Milford, and K. D. McDonald-Maier, "Memorable maps: A framework for re-defining places in visual place recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 12, pp. 7355–7369, Dec. 2021.

[81] M. Zaffar *et al.*, "VPR-bench: An open-source visual place recognition evaluation framework with quantifiable viewpoint and appearance change," *Int. J. Comput. Vis.*, vol. 129, no. 7, pp. 2136–2174, 2021.

[82] M. Zaffar, A. Khaliq, S. Ehsan, M. Milford, K. Alexis, and K. D. McDonald-Maier, "Are state-of-the-art visual place recognition techniques any good for aerial robotics?," in *Proc. IEEE ICRA Workshop Aerial Robot.*, 2019.

[83] M. Zaffar, A. Khaliq, S. Ehsan, M. Milford, and K. McDonald-Maier, "Levelling the playing field: A comprehensive comparison of visual place recognition approaches under changing conditions," in *Proc. IEEE ICRA Workshop Dataset Gener. Benchmarking SLAM Algorithms Robot. VR/AR*, 2019.

[84] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, 1452–1464, Jun. 2018.

[85] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.

[86] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Representations*, 2017.

[87] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 413–422.

Bruno Ferrarini received the M.Sc. degrees in electronic engineering and ICT engineering from the University of Parma, Parma, Italy, in 2002 and 2006, respectively, and the M.Sc. degree (by dissertation) in computer and electronic systems, in 2016, from the University of Essex, Colchester, U.K., where he is currently working toward the Ph.D. degree in computer science.

His research interests include machine learning, local image feature extraction, visual place recognition, and binary neural networks.

Michael J. Milford (Senior Member, IEEE) received the Bachelor of Mechanical and Space Engineering degree and the Ph.D. degree in electrical engineering from The University of Queensland, Brisbane, QLD, Australia.

He was a Research Fellow on the Thinking Systems Project with Queensland Brain Institute, in 2010. Since 2010, he has been a Lecturer with the Queensland University of Technology (QUT), Brisbane, QLD, Australia, where he is currently an Associate Professor and an Australian Research Council Future Fellow and the Chief Investigator of the Australian Centre of Excellence for Robotic Vision. His research interests include navigation across the fields of robotics, neuroscience, and computer vision.

Dr. Milford was a recipient of an inaugural Australian Research Council Discovery Early Career Research Award in 2012. He has been a Microsoft Research Faculty Fellow since 2013.

Klaus D. McDonald-Maier (Senior Member, IEEE) received the Dipl.-Ing. degree in electrical engineering from the University of Ulm, Ulm, Germany, the M.S. degree in electrical engineering from CPE Lyon, Villeurbanne, France, in 1995, and the Ph.D. degree in computer science from Friedrich Schiller University Jena, Jena, Germany, in 1999.

He was a Systems Architect on reusable microcontroller cores and modules with Infineon Technologies AG's Cores and Modules Division, Munich, Germany, and a Lecturer of Electronics Engineering with the University of Kent, Canterbury, U.K. Since 2005, he has been with the University of Essex, Colchester, U.K., where he is currently a Professor with the School of Computer Science and Electronic Engineering. His research interests include embedded systems and system-on-a-chip design, security, development support and technology, parallel and energy efficient architectures, and the application of soft computing and image processing techniques for real-world problems.

Dr. McDonald-Maier is a member of the Verband der Elektrotechnik Elektronik Informationstechnik and the British Computer Society, and a Fellow of the Institution of Engineering and Technology.

Shoaib Ehsan received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Taxila, Pakistan, in 2003, and the Ph.D. degree in computing and electronic systems, with a focus on computer vision, from the University of Essex, Colchester, U.K., in 2012.

He has extensive industrial and academic experience in the areas of embedded systems, embedded software design, computer vision, and image processing. His research interests include intrusion detection for embedded systems, local feature detection and description techniques, image feature matching, and performance analysis of vision systems.

Dr. Ehsan was a recipient of the University of Essex Post Graduate Research Scholarship and the Overseas Research Student Scholarship. He was a winner of the prestigious Sullivan Doctoral Thesis Prize by the British Machine Vision Association.