Version of Record: https://www.sciencedirect.com/science/article/pii/S0010448520301779
Manuscript_667f50f3c5dbe35157f876fb2280f824

A Hardware-adaptive Deep Feature Matching Pipeline for Real-time 3D Reconstruction

Shuai Zheng, Yabin Wang

School of Software Engineering, Xi'an Jiaotong University, China

Baotong Li

School of Mechanical Engineering, Xi'an Jiaotong University, China

Xin Li

School of Electrical Engineering and Computer Science, Louisiana State University, USA

Abstract

This paper presents a hardware-adaptive feature modeling framework to automatically generate and optimize deep neural networks to support real-time feature extraction and matching on a given hardware platform. This framework consists of a deep feature extraction and matching pipeline and a neural architecture search scheme, with which deep neural networks can be automatically generated and optimized according to given hardware to achieve reliable real-time feature matching. Built on our feature matching approach, we also developed a real-time 3D scene reconstruction pipeline that could run adaptively on hardware with different computational performance. We designed experiments to validate the proposed matching and reconstruction pipelines on hardware platforms with different performance. The results demonstrated our algorithms' effectiveness on both matching and reconstruction tasks.

Keywords: Deep Feature Matching, Neural Architecture Search, Real-time 3D Reconstruction.

1. Introduction

3D reconstruction is an important technique in the field of computer-aided design and engineering, because being able to easily acquire 3D shapes or models from the real world can greatly accelerate the design and analysis [1, 2, 3, 4, 5]. A key technique in 3D reconstruction is *feature extraction and matching*, which helps register images taken from different view angles and then infer depth and geometric structures of the captured scenes/objects. An accurate and robust feature extraction and matching pipeline is critical towards effective 3D reconstruction, especially when dealing with challenging free-conditioned environments such as those scenes that have limited field of views [6], low scanning rate [7], or are sparsely sampled [8]. Besides accuracy, feature modeling and matching often need to be performed in real-time for interactive re-scanning and refinement. But this is often non-trivial if the given portable platform (due to cost or mobility constraint) has low computational performance. Often times, trade-offs need be made between the efficiency and accuracy of matching and reconstruction algorithms. For example, on high-performance workstations, more sophisticated algorithms or deeper neural networks can be used to extract more detailed feature information, but on a smaller lower-cost laptop, simpler algorithms or networks should be adopted to accommodate the efficiency requirements.

A feature matching pipeline typically consists of two main steps: *keypoint detection* and *feature description*. Classical keypoint detection and feature description pipelines include SIFT [9], SURF [10], FAST [11], etc., which have been widely adopted in many matching, reconstruction, and visual SLAM (simultaneous localization and mapping) tasks. Recently, deep learning based keypoint detection [12, 13], feature description [14, 15], and the combined detection-description pipelines [16, 17, 18, 19, 20] have been explored and achieved the state-of-the-art performance. However, most of these learning-based pipelines rely on a pre-designed neural network architecture, which might work well on some platforms, but not on some others. Specifically, a network designed and trained on a workstation with high-end GPUs could undergo a significant performance drop when it is deployed on a computationally limited device such as a laptop. Our goal is to develop a hardware-adaptive pipeline that can automatically optimize the neural network architecture according to the given hardware's performance.

Inspired by the recent study in neural architecture search (NAS) research in image classification and recognition tasks [21, 22, 23, 24], we developed a novel learning-based feature matching pipeline with an NAS optimization scheme to support the hardware-adaptive interactive feature matching. On a given hardware platform, a neural network with restricted latency will be generated to support on-the-fly 3D reconstruction. The main contributions of this work can be summarized as:

1. We developed a first automatic hardware-adaptive deep feature modeling and matching framework that can generate deep neural networks for keypoint detection and feature description according to given hardware. We formulated this as a latency-constrained neural network architecture optimization problem, which turns out to be effective and successfully makes the pipeline hardware-adaptive. This framework can help avoid tedious manual network redesign when deployed on new given hardware platforms.
2. Built upon this deep feature matching pipeline, we developed a 3D reconstruction system to reconstruct 3D scenes, and validated that it can achieve real-time performance in

*Corresponding author
Email address: xinli@lsu.edu (Xin Li)

three hardware platforms with different computational performance.

3. Compared with existing feature matching algorithms, the proposed pipeline achieved the state-of-the-art performance in accuracy and efficiency, especially on low-performance hardware platforms.

2. Related Work

This section discusses related work in the following three categories: (1) learning based feature extraction, (2) Neural Architecture Search, and (3) 3D reconstruction based on sparse feature correspondence.

2.1. Learning-based Feature Extraction

Learning-based keypoint detectors. Deep learning techniques boosts the development of new effective keypoint detectors. FAST [25] and ORB [26] use machine learning approaches to speed up the process of corner detection. TILDE [27] learns from pre-aligned images of the same scene at different illumination conditions. Although being trained with the assistance from SIFT, TILDE can still identify keypoints missed by SIFT, and outperforms SIFT. Quad-Network [12] is trained unsupervisedly with the help of a ranking loss. Zhang et al. [28] combined this ranking loss with a "Peakedness" loss and produces a more repeatable detector. Lenc et al. [13] proposed to train a keypoint detector directly from the covariant constraint. Orientations of keypoints can also be estimated through learning [29] to facilitate the subsequent keypoint matching.

Learning-based feature descriptors. Learning effective feature description is critical in establishing correspondence and matching between data. DeepDesc [30] integrates a Siamese network with the MatchNet [31] and Deepcompare [15], to learn nonlinear distance matrices for feature matching. A series of recent studies have explored more advanced architectures and triplet-based deep metric learning formulations, including UCN [32], TFeat [33], GLoss [34], L2-Net [35] and HardNet [36] for feature learning. These methods focus on designing better loss functions, but still using the same network architecture proposed in L2-Net [35].

Joint detector-descriptor learning. LIFT [16] was probably the first work aiming to build an end-to-end feature detection and learning pipeline. In this work, keypoints are detected and cropped regions are then fed to a second network to estimate the orientation before going throughout a third network to perform description. However, in the training phase, it relies on the output of SIFT detector to initialize, thus it only partially achieves this goal. SuperPoint [17] trains a convolutional neural network that consists of a single shared encoder and two separate decoders for keypoint detection and feature description, respectively. Synthetic shapes are used to generate images for detector's pre-training, and synthetic homographic transformations are used to produce image pairs for detector's fine tuning. Based on Q-learning, LF-Net [18] uses a Siamese architecture to train the entire network without the help of hand-craft method. RF-Net [19] modifies the LF-Net architecture to construct more effective receptive feature maps, and introduces a neighbor mask loss term to facilitate training patch selection for more stable descriptor training.

While recent research on keypoint detection and feature description learning has achieved remarkable results, these deep networks are pre-designed and tuned, and their performance may not be stable when deployed on different hardware platforms.

2.2. Neural Architecture Search

The majority of current deep neural networks are developed manually, which are often time-consuming and difficult to reproduce/tune [37]. Neural Architecture Search (NAS) is a new trend in efficient network design that searches the optimal neural network from a hierarchical cell-wise search space automatically. Earlier studies of NAS focused on the cell level structure search, and the same cell is used repeatedly in the entire neural network. Nasnet [21] uses block-level hierarchical search space to increase the diversity of the searched neural network, in which a network is formed by different cells and each cell is formed through different blocks. Weight sharing is often adopted in NAS to reduce computational cost. A "SuperNet" that contains different possible network topology architectures is built and then pruned to obtain an optimal architecture through iterative training.

NAS has been successfully applied in some recent image feature classification tasks, especially since the development of gradient-based NAS frameworks, in which the search reduces to a numerical optimization problem. DARTS [22] is probably the first gradient based end-to-end NAS framework, in which a joint optimization of the architecture and its weights can be conducted through a continuous relaxation scheme. It produces more effective convolutional architectures in image classification than previous NAS algorithms. GDAS [38] is also a Gradient-based search method using Differentiable Architecture Sampler to do image classification. Built upon DARTS, a main improvement of GDAS that it only update one sub-graph each time rather than the entire SuperNet. So it is more efficient and can avoid the mutual inhibition caused by updating all the operations together. P-DARTS [39] tries to solve the "depth gap" problem in DARTS, which uses a deeper network but fewer operations in every cell. PC-DARTS [40] designs a channel sampling technique to further reduce memory and computation. FBNet [24] uses gradient-based methods to search higher accuracy neural networks for image classification, considering the hardware performance as a constraint of the optimization.

The above development of NAS on the image feature classification tasks are gradient based, but the searching aims at finding only one optimal block for each cell. So, the final resultant framework is sequential. Selecting only one block with highest weight and abandoning other blocks makes the optimization an integer problem, whose numerical optimization is in fact harder than making the optimization a true continuous problem. In this work, we remove this integer constraint to make the architecture design more flexible and optimization more efficient.

2.3. 3D Reconstruction

We briefly review closely related work on feature extraction and matching algorithms in 3D reconstruction, and refer the readers to a detailed survey of 3D reconstruction algorithms [41].

3D reconstruction approaches can be categorized into *dense matching* and *feature-based matching* methods. In *dense matching* methods, all pixels between two frames are analyzed based on their geometric and/or photometric characters. Representative dense matching methods include ElasticFusion [42] and RGB-DTAM [43]. To achieve pixelwise dense matching, volumetric representations are often adopted. These approaches often require a powerful graphic hardware to carry the parallel computation, and hence, may not run on hardware with low computational performance. Reconstructions using *feature-based matching* first establish coarse correspondences between features in different frames, and then estimate inter-frame alignment based

on these correspondences. The matching is based on the keypoint descriptors and geometric constraints. A state-of-the-art performance is achieved through ORB-SLAM2 [2]. The ORB descriptor is a binary vector allowing high performance matching. GCNv2 [44] replaced the detector and descriptor of ORB-SLAM2 by neural networks, and it achieves a comparable accuracy as ORB-SLAM2. However, most of these existing 3D reconstruction algorithms were designed and tested on a single hardware platform. How to design a flexible framework that can perform real-time feature matching and 3D reconstruction on different hardware platforms adaptively has been little explored.

3. Method

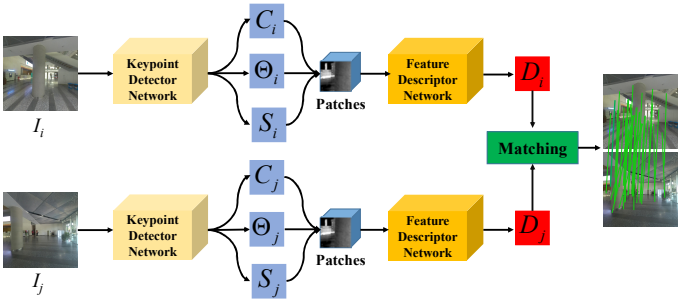


Figure 1: Our Automatic Feature Extraction and Matching Pipeline. For keypoint extraction, the detector network generates a score map C, a dense orientation estimation Θ, and a scale map S. Image patches around the chosen keypoints are cropped and fed to the descriptor network to compute descriptor D. With these matching between two images can be computed.

Recently, learning based feature matching methods [17, 18, 19, 20] have outperformed traditional hand-designed methods and achieved the state-of-the-art performance. But their accuracy gain comes with the price of deploying complicated neural networks on high-end GPUs. When the system needs to be ported onto machines with lower computational capability for real-time applications, these deep feature matching systems' performance often drops significantly, and one needs to re-design/re-train the networks on the given platform which is not only tedious but also difficult. Our goal is to build a flexible framework that adapts to the given platform, and balances the accuracy and efficiency if they cannot be achieved simultaneously. Following the most widely adopted two-phase frameworks, we propose to design and then optimize our matching pipeline consisting of the *keypoint detection* and *feature description* modules. Our pipeline is illustrated in Figure 1. Given an image pair I_i and I_j, our *keypoint detection* network generates the pixel-wise feature saliency score maps C_i and C_j, orientation maps Θ_i and Θ_j, and scale maps S_i and S_j, respectively. Keypoints are extracted from the score maps. Then, patches around keypoints are cropped from I_i and I_j, and fed to the *feature description network* to produce patch descriptors D_i and D_j, with which the feature correspondence can be obtained via nearest neighbors in the descriptor space. We explain our design of the *keypoint detection* network in Section 3.1 and *feature description* network in Section 3.2, their composition and training in an end-to-end pipeline in Section 3.3, and the utilization of this pipeline for real-time 3D reconstruction in Section 3.4.

3.1. Hardware-adaptive Keypoint Detection

We developed a *keypoint detection* module using LF-Net [18] as the baseline, but we make this neural network adaptively adjustable. LF-Net uses a convolutional network to detect keypoint locations on images from their learned corresponding scale and rotational maps. It achieves a good balance between efficiency and accuracy, and hence, is a suitable starting point. However, the network architecture in LF-Net is fixed and computationally intensive, it is not suitable for some hardware with low performance. Therefore, we make the number of convolutional blocks adaptively adjustable according to given hardware.

The proposed keypoint detector network is shown in Figure 2. Given an input image, we first use N_{det} convolutional blocks B to extract feature maps F, which are used to build a multi-scale feature pyramid to get multi-scale responses. N_{det} is selected according to hardware performance. Similar to LF-Net, we compute a score map, a scale map, and an orientation map from the feature pyramids.

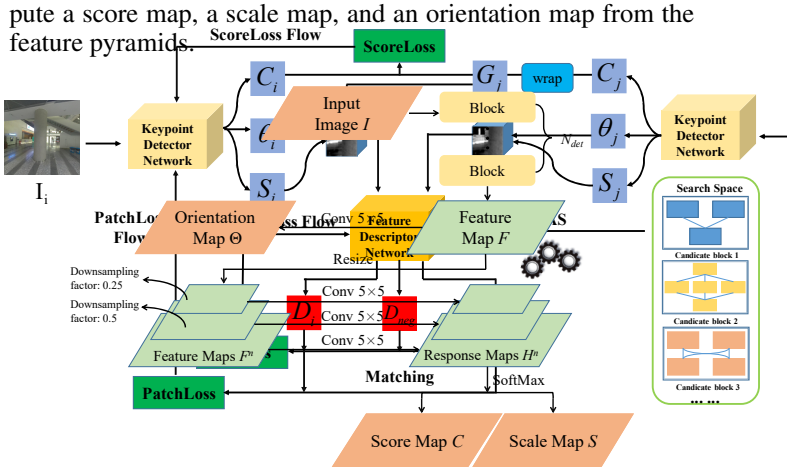


Figure 2: Our keypoint detection network that estimates keypoint locations, scales, and orientations. N_{det} blocks are used to generate a feature map F from an input image I. A $5\times5\times2$ convolutional layer is applied on the feature map to obtain the orientation map Θ. To obtain a multi-scale pyramid, the input feature map is downsampled by a factor of 0.5 and 0.25. Feature map pyramids are fed into a 5×5 convolutional layer to generate multi-scale response maps H^n, from which score maps C and scale map S are obtained through SoftMax.

To determine N_{det} according to hardware performance, we first calculate the input data size for each block in the detector network, so the latency of each block T_B on a specific hardware can be estimated. Then, the total latency from concatenating N_{det} blocks should be restricted. For example, suppose on a real-time application such as 3D reconstruction that should run at a 10-15fps, the entire matching pipeline needs to finish within 100ms. We denote this time as a *target latency* T_{target}. Based on our experiments and observation, in most existing learning-based matching pipelines, the keypoint detection and feature description learning modules consume about 30% and 70% time, respectively. Therefore, we also use about 30% of the *target latency* to restrict the latency of keypoint detection. This is formulated as

$$N_{det} = \frac{30\% \times T_{target} - T_{other}}{T_B}, \tag{1}$$

where T_B is the latency (in milliseconds) consumed by each block, and T_{other} is latency from other network operations.

From these convolutional blocks, a tensor of feature maps, F, can be obtained. Then a multi-scale pyramid of feature maps is built to get scale-space responses by downsampling F for N ($N = 3$ in our implementation) times with a factor of 0.5. These feature maps are convolved by a 5×5 filter to get N response maps H^n ($1 < n < N$). We resize each H^n back to the original image size to obtain pixel-wise score maps.

To increase the saliency of keypoints, we perform a differentiable form of non-maximum suppression by applying a softmax

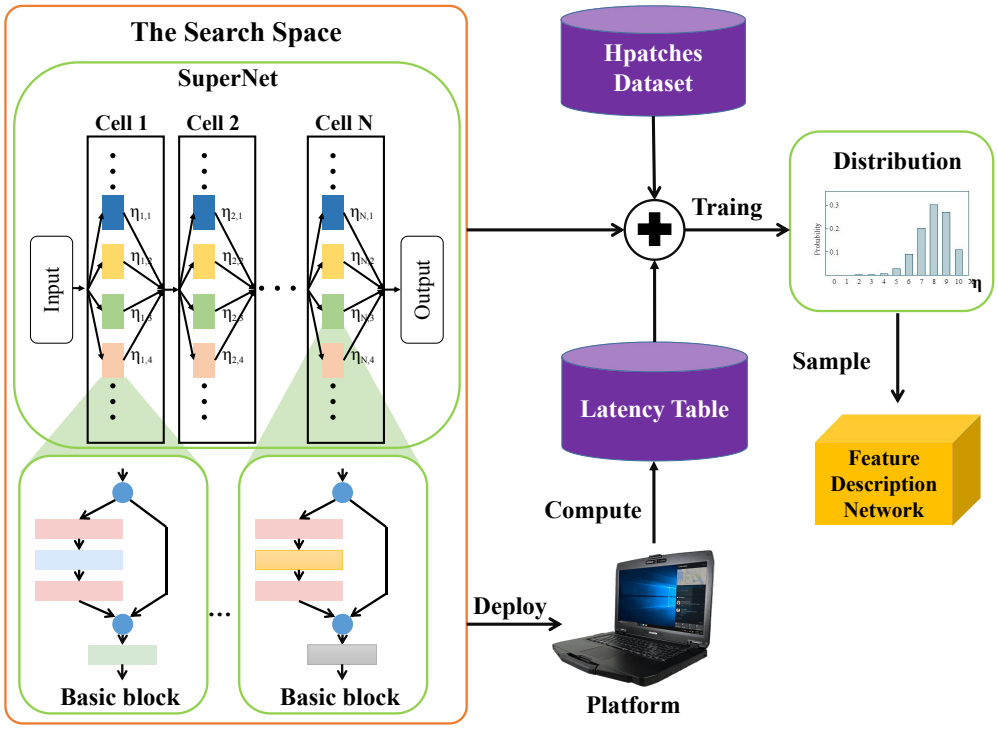


Figure 3: Our NAS Framework. The Search Space contains candidate convolutional blocks, and a SuperNet is created using these blocks. Latency of the candidate convolutional blocks is computed according to the SuperNet architecture. The searching process optimizes the SuperNet weights and architecture parameters η to minimize the loss function $L(a, w_a)$, yielding the final architecture $a \in A$.

operator over a 15×15 window in a convolutional manner. Then we merge all these sharper response maps to get the final score map C by a softmax operation with a $1 \times 1 \times N$ window.

$$C = \sum H^n \odot softmax_n(H^n), \tag{2}$$

where operator $\odot$ is the Hadamard product.

The scale of each layer in the response maps is defined as the size of the corresponding layer's receptive field. Each pixel in the scale map S is defined as the scale of the layer with the largest intensity in the response maps. By applying a $5 \times 5 \times 2$ convolutional layer on the feature map, we can obtain the orientation map Θ. The output of orientation map Θ consists of two channels represented by *sine* and *cosine* of every pixels. In this way, the final orientation can be described by *arctan*, which indicates the rotational angle between patches from the two images.

The output of the keypoint detection network is a set of keypoints $\{v_i = (x_i, y_i)\}$, their scales s_i, and orientations θ_i. The keypoints are the top-K responses detected from the response score map S.

3.2. Optimizing Feature Description

We designed our feature description module based on L2Net [35], but searched the neural network according to the hardware computational performance. L2Net has been widely adopted in many recent feature matching frameworks such as HardNet [36] and RF-Net [19]. It uses seven convolutional layers to produce 128-dimensional descriptors, and has been shown effective in many recent feature matching pipelines. However, the fixed L2Net structure is not adaptive when the hardware platform changes. Furthermore, compared with keypoint detection, the descriptor learning plays a more important role in the entire feature matching pipeline. Therefore, we performed an optimization on the feature description network using differentiable neural architecture search (DNAS) [22]. We also designed a new loss term to guide the network architecture search and optimize feature description with constrained latency.

3.2.1. Network Design

To optimize the neural network design for feature description, we perform an NAS to search for an efficient and effective network from various network combination. The search is usually very large and a brute-force enumeration and training of all the potential sub networks is prohibitive. A recent state-of-the-art NAS framework is FBNet [24], in which the search space A is initialized as a neural network composed by several sequentially connected *cells*. This network is also called *SuperNet*, and its structure is fixed. FBNet optimizes the selection of blocks in this fixed structure, to enhance the performance of an image classification network.

In this work, we revised the framework of FBNet to make the pipeline more suitable for our real-time task. The two main modifications we made to the FBNet pipeline are that: (1) In each layer, our search network contains multiple blocks rather than a single block (FBNet). This allows us to construct more complicated network structures rather than just simple, sequential ones. (2) We changed the latency term to a hard constraint, with which the efficiency of our pipeline in real-time applications can be guaranteed.

Our developed neural architecture search framework is illustrated in Figure 3. Each cell is allowed to have several parallelly allocated candidate blocks. The input of each cell is a feature map, fed into these candidate blocks; the cell outputs a weighted summation of each block's outputs. We denote these weight as η. Each candidate block contains several convolutional layers,

whose network parameters are denoted as w. During the training stage, η and w are simultaneously learned. When converged, blocks with $\eta < 0.1$ are trimmed. The final pipeline, referred to as the optimal sub network a, may contain different cells.

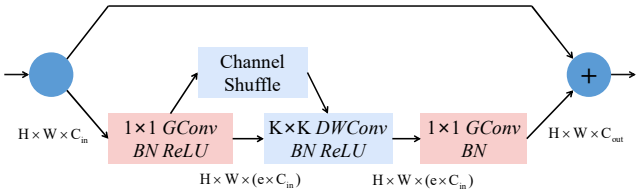


Figure 4: The basic convolutional block. C_{in}, C_{out} indicate the number of input and output channels ; H and W are the height and width of the input; K is the kernel size of the depthwise convolution (DWConv); e is the adjustment magnification of the input channel; GConv and BN are group convolution and batch normalization.

Candidate Blocks. The design of basic block candidates is inspired by the lightweight network [45, 46] and the residual network [47]. As illustrated in Figure 4, it contains a 1×1 pointwise convolution, a $K \times K$ depthwise convolution, and a 1×1 pointwise convolution. Batchnorm and Relu operations follow each convolution, serving as the normalization and activation functions. The channel adjustment magnification is expressed as expansion e. The final output of the basic block is the sum of the result and the input. The Channel Shuffle (CS) is an optional module, which mixes the information between channels to obtain better accuracy to the neural network. A few such blocks can replace multiple convolutional layers to achieve similar accuracy, but with less computation and better efficiency. These blocks are generated by a variation of k and e defined in the basic block. To search for optimal cells, we design 9 types of canonical candidate blocks, as listed in Table. 1. We denote a block of $k = 3, e = 1$ as $k3\_e1$. In fact, the blocks adopted in the keypoint detection network (Section 3.1) is this basic $k3\_e1$ block. Here the block type of "skip" means skipping this entire block.

Table 1: Candidate Blocks

Block type	Expansion	Kernal	CS
k3_e1	1	3	No
k3_e1_s2	1	3	Yes
k3_e3	3	3	No
k3_e6	6	3	No
k5_e1	1	5	No
k5_e1_s2	1	5	Yes
k5_e3	3	5	No
k5_e6	6	5	No
skip	-	-	-

Layer-wise SuperNet. The SuperNet we design is also layer-wise. Each layer is a cell which contains several different candidate blocks. The whole SuperNet is composed by N_{des} sequentially connected *cells*. The search procedure will select and combine optimal candidate blocks to form each cell in the SuperNet. A SuperNet architecture (with $N_{des} = 6$ layers of cells) is shown in Table 2. The first and last cells of the network are determined in order to generate fixed length descriptors, while other cells are defined through space searching.

Table 2: Architecture of the layer-wise search space. Column-"Block" denotes the block type. Column-"filters" denotes the filter number of a block. Column-"stride" denotes the stride of the first block in a stage. The final output is a batch of 128 dimension vectors.

Input shape	Cell	filters	stride
$32^2 \times 1$	3×3Conv	32	1
$32^2 \times 32$	Cell 1	32	2
$16^2 \times 32$	Cell 2	64	1
$16^2 \times 64$	Cell 3	64	2
$8^2 \times 64$	Cell 4	128	1
$8^2 \times 128$	Cell 5	128	2
$4^2 \times 1$	Cell 6	128	1
$4^2 \times 1$	4×4Conv	-	-

3.2.2. *Continuous Optimization of Weights*

Classical NAS approaches combinatorially select blocks with a binary 0-1 mask and re-initialize w in the next iteration, which is computationally expensive and sometimes unstable. Recently, FBNet [24] suggests a differentiable neural architecture search framework that uses gradient-based methods to optimize the network architectures, and it achieves great performance. In each iteration, network parameters w are updated based on w from the previous iteration, which significantly reduces the searching time. Therefore, here we followed FBNet to search and optimize our feature description neural network in this continuous way. Furthermore, when the optimization is finished, unlike FBNet that only selects a single block, we kept multiple blocks.

The searching process in classical NAS frameworks is formulated as an integer optimization problem. The input for cell $l+1$ can be obtained by

$$x_{l+1} = \sum_i m_{l,i} \cdot b_{l,i}(x_l), \tag{3}$$

$$\begin{cases} \hat{i} &= \arg\max_i(\eta_{l,i}) \\ m_{l,i=\hat{i}} &= 1 \\ m_{l,i\neq\hat{i}} &= 0 \end{cases} \tag{4}$$

where $m_{l,i}$ is an 0-1 integer, x_l denotes the input for cell l, and $b_{l,i}$ denotes the operation in cell l and block i.

Because $m_{l,i}$ is not differentiable, the optimization problem cannot be solved with gradient descent. To convert the discrete parameters $m_{l,i}$ into continuous variables for stochastic gradient descent (SGD), a Gumbel Softmax function is used to replace Equation. 4,

$$m_{l,i} = \text{GumbelSoftmax}(\eta_{l,i}|\eta_l) = \frac{\exp[(\eta_{l,i} + g_{l,i})/\tau]}{\sum_i \exp[(\eta_{l,i} + g_{l,i})/\tau]}. \tag{5}$$

Here $g_{l,i}$ is a random noise for cell l and block i, following the Gumbel distribution. The Gumbel Softmax function is controlled by a temperature parameter τ. When τ approaches 0, $m_{l,i}$ approximates the discrete categorical sampling, and when τ becomes larger, $m_{l,i}$ becomes a continuous random variable. For any τ value, $m_{l,i}$ is differentiable with respect to parameter $\eta_{l,i}$.

Figure. 5 compares the data flow before and after using GumbelSoftmax. When GumbelSoftmax is not adopted, as is shown on the top, the $m_{l,i}$ for each block is either 0 or 1, where its corresponding block is rendered in white and black. On the other hand, when GumbelSoftmax function is used, as rendered on the bottom, blocks are rendered in gray scale, indicating its $m_{l,i}$ is between 0 and 1, where a lighter block has a larger $m_{l,i}$.

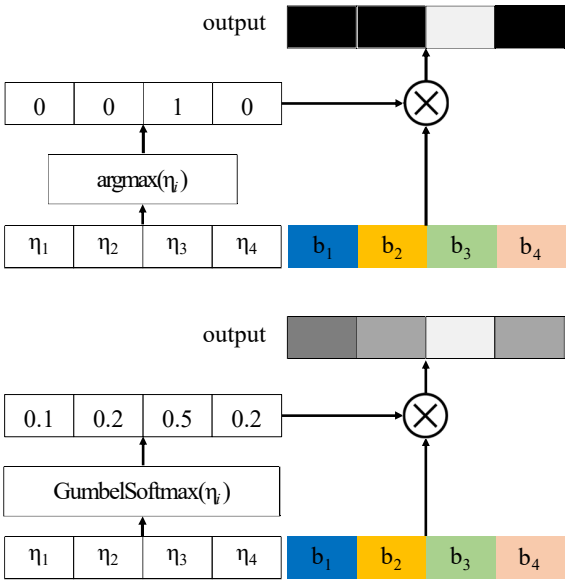


Figure 5: Data flow before and after using GumbelSoftmax operation. When GumbelSoftmax is not adopted, as is shown on the top, the $m_{l,i}$ for each block is either 0 or 1, where its corresponding block is rendered in white and black. On the other hand when GumbelSoftmax function is used, as rendered on the bottom, blocks are rendered in gray scale, indicating its $m_{l,i}$ is between 0 and 1, where a lighter rendered block refers to a larger $m_{l,i}$.

In this way, the input for cell $l+1$ can be obtained by

$$x_{l+1} = \sum_i \text{GumbelSoftmax}(\eta_{l,i}|\eta_l) \cdot b_{l,i}(x_l), \tag{6}$$

where x_l denotes the input for cell l, and $b_{l,i}$ denotes the convolutional operations in cell l and block i.

After searching, we select blocks with $m_{l,i} > \varepsilon$ from each layer to construct the final feature description network. Here ε is a threshold, and through experiments, we found setting it to 0.1 works well.

3.2.3. Loss Function

To ensure the delay of our entire network to be smaller than the target latency, we shall evaluate the running time of each selected sub network a on the target device. However, measuring the latency of a during the searching procedure is computationally expensive and often prohibitive in practice. Therefore, we first deploy each block in the search space on the target device, measure the running time, and record it in a lookup table. This lookup table helps estimate the total latency of a by the aforementioned mask variable $m_{l,i}$ in a differentiable form

$$T(a) = \sum_l \sum_i m_{l,i} \cdot T(b_{l,i}), \tag{7}$$

where $b_{l,i}$ is a selected block i at cell l from a.

Furthermore, we evaluate the distance from the latency of the current sub network $T(a)$ to the target latency T^{des}_{target}. Since we introduce the mask variable $m_{l,i}$ to make the optimization differentiable, the latency calculated is therefore not accurate and usually larger than the true latency. Thus, we propose a new way to represent the current sampled latency of the network, which we call a "softly sampled latency". Inspired by the soft-NMS algorithm, we obtain the maximum softened probability for each cell of m. The soft-NMS operation is that for every blocks i in cell l, the probability $m_{l,i}$ is suppressed when it is small, and is enlarged when it is large:

$$\text{soft}\_m_{l,i} = \frac{\exp(m_{l,i} - \max(m_l))}{\sum \exp(m_{l,i} - \max(m_l)}. \tag{8}$$

Then we can get the soft sample latency $\hat{T}$ by

$$\hat{T} = \sum_l \sum_i \text{soft}\_m_{l,i} \times T(b_{l,i}). \tag{9}$$

The total latency loss function is therefore written as

$$L_{lat} = \sum_l \sum_i m_{l,i} \cdot T(b_{l,i}) \times (\frac{\hat{T} - T^{des}_{target}}{T^{des}_{target}})^2. \tag{10}$$

The loss we use to train the feature description network is based on the HardNet loss[36]. It maximizes the distance between the closest positive and closest negative example in the batch. The description loss L_{des} is formulated as:

$$L_{des}(D_{pos}, D_{ng}) = \frac{1}{K} \sum \max(0, 1 + D_{pos} - D_{ng}) \tag{11}$$

where

$$D_{pos} = d(D^k_i, D^k_j) \tag{12}$$

$$D_{ng} = min(d(D^k_i, D^k_j), d(D^m_i, D^k_j)) \tag{13}$$

D^n_j is the closest non-matching descriptor to D^k_i and D^m_i is the closest non-matching descriptor to D^k_i. The descriptors' distance is the Euclidean distance between the centroids of the two patches.

The final loss function is defined as:

$$L = \alpha Loss_{acc} + \beta Loss_{lat}, \tag{14}$$

where α and β are weight parameters. Through experiments we found that simply setting $\alpha = 1.0$ and $\beta = 1.0$ works well.

3.3. End-to-end Training

Many recent deep frameworks [18, 19] that jointly learn keypoint detection and feature description adopt three main loss functions: score loss, patch loss, and description loss. The *score loss* measures the repeatability and reliability of the detected keypoints on different illumination and viewpoint settings. The *patch loss* measures the description difference between regions surrounding corresponding keypoints in two images. The *description loss* measures the difference between a closest positive pair and the similarity between a closest negative pair.

In the end-to-end training process, the Keypoint detector is not separately trained through supervised learning, but trained together with the subsequent feature description. A point is considered as a keypoint if (1) it has good reliability, i.e., not sensitive to viewpoint and illumination changes, and (2) it is unique and unambiguous, i.e., it can have discriminative description that is different from that of other keypoints. Therefore, keypoints are defined and trained using the score loss and patch loss together with the descriptors. To support this end-to-end training, we use training data (to elaborate in Section 4.1) that contain many image pairs with known homography transformations. These data can support the aforementioned training. The training process is designed as illustrated in Figure 6.

Score loss. The score loss is designed to train and evaluate the extraction of score map from the keypoint detection neural

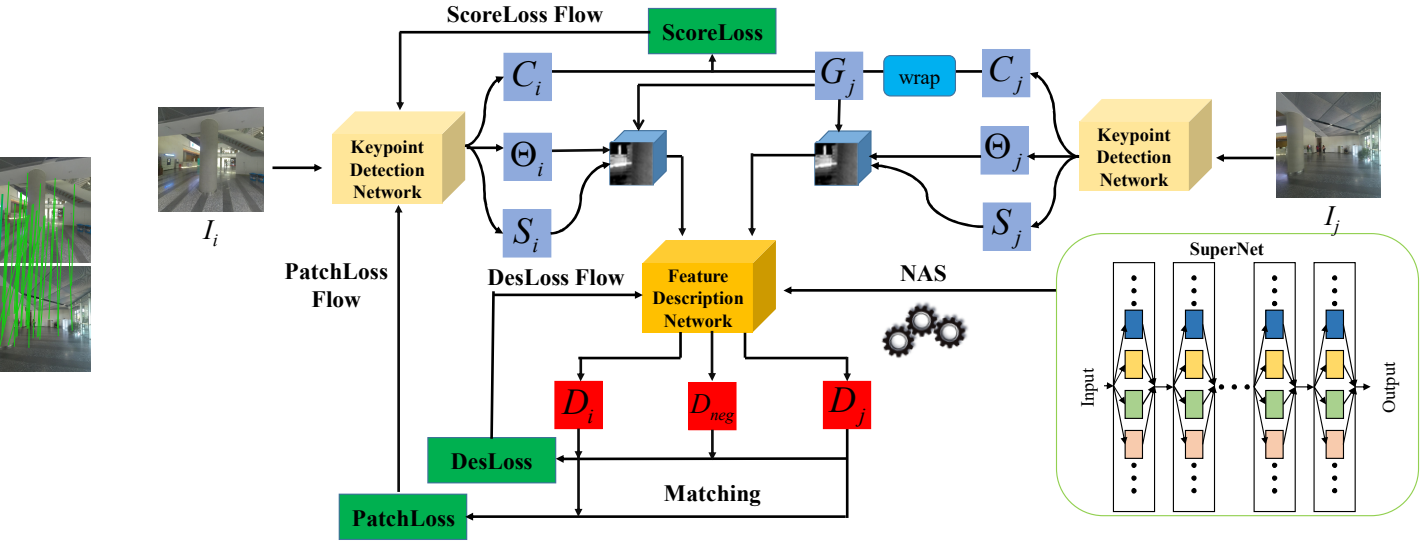


Figure 6: Automatic Training Pipeline. In the End-to-End training pipeline, we firstly use the proposed NAS method to search the feature description network architecture for the target hardware platform. Then we combine it with keypoint detection network to do end-to-end training. In the training process, we feed an image pair (I_i and I_j) to get corresponding score maps C_i and C_j. Then we process C_j to produce ground truth G_{ji}, which is considered as the ground truth for C_i. Furthermore, the gradient of the L_{score} loss function is calculated with C_i updated by the left keypoint detection network. Similarly, L_{patch} and L_{des} are calculated and back propagated to the networks.

network, to ensure that the score maps C_i and C_j from images I_i and I_j have the same score on corresponding keypoint locations. First, feed an image pair (I_i and I_j) into the network to get corresponding score maps C_i and C_j. Second, transform the viewpoint of score map C_j into that of C_i, by a perspective transformation operation w. Then, process C_j by a Non-Maximum-Suppression (NMS) to make the points of interest sparse and sharp. After that, select top K (in signal stren[illegible]s as ke[illegible] denote this operation as κ. Finally, [illegible] a clean[illegible]h score map G_{ji} by placing a Gaussian kernel g with standard deviation $\sigma = 0.5$ at those locations. T[illegible]oss is d[illegible] mean-square error (MSE) betwee[illegible]$_{ji}$. Th[illegible]rations are expressed in Equation. 15, and the score loss $L_{[illegible]}$ is defined in Equation. 16.

$$G_{ji} = [illegible] \quad (15)$$

$$L_{score}(C_i, C_j) = \|C_i - G_{ji}\|^2 \quad (16)$$

Patch loss. Keypoint orientations and sca[illegible] important in accurate feature extr[illegible] keypoint orientations and scales need to be consisten[illegible]t view[illegible]ination conditions. We extract a patch surrounding the k-th keypoint's θ and s from image I, and denote it as $I(x_i^k, y_i^k, s_i^k, \theta_i^k)$. The patch is fed to the description network F to get its description D_i^k:

$$D_i^k = F(I(x_i^k, y_i^k, s_i^k, \theta_i^k)) \quad (17)$$

The patch loss L_{patch} can then be defined by

$$L_{patch}(D_i^k, D_j^k) = \frac{1}{k}\sum_{1}^{k} d(D_i^k, D_j^k) \quad (18)$$

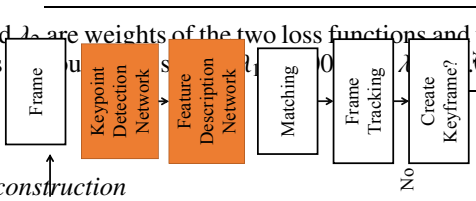
where $d(x, y) = \|x - y\|_2$ is the distance between x and y.

Description loss. The aforementioned description loss L_{des} in Equation.11 is used to train the description network.

To train the keypoint detection network, L_{score} and L_{patch} are combined as L_{det}:

$$L_{det} = \lambda_1 L_{score} + \lambda_2 L_{patch}, \quad (19)$$

where λ_1 and λ_2 are weights of the two loss functions and through experiments [illegible] works well.



3.4. 3D Reconstruction

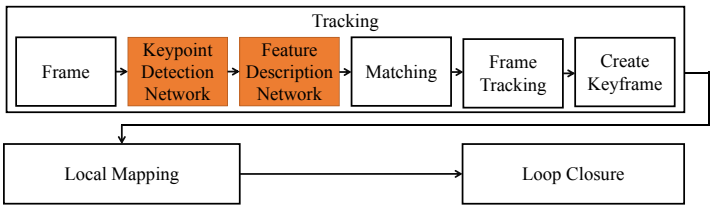


Figure 7: The flowchart of our 3D reconstruction system. We just feed the input image into the keypoint detection network, which computes keypoint locations, orientations and scales. After that we use these information to extract corresponding image patches. By feeding these image patches into the feature description network, descriptors for each keypoints are obtained. We use a standard nearest neighbor search algorithm to create correspondences between keypoints. The rest of the 3D reconstruction system is kept as the same as ORB-SLAM.

We designed a 3D reconstruction system based on the widely adopted ORB-SLAM2 [2] system, which is regarded as a state-of-the-art in 3D reconstruction tasks. In the pipeline of ORB-SLAM2, we replaced the feature extraction module with our keypoint detection and feature description networks, as the orange boxes shown in Figure 7. Our pipeline takes consecutive image frames to compute keypoints and their descriptors, then uses a standard nearest neighbor search algorithm to compute correspondence between keypoints.

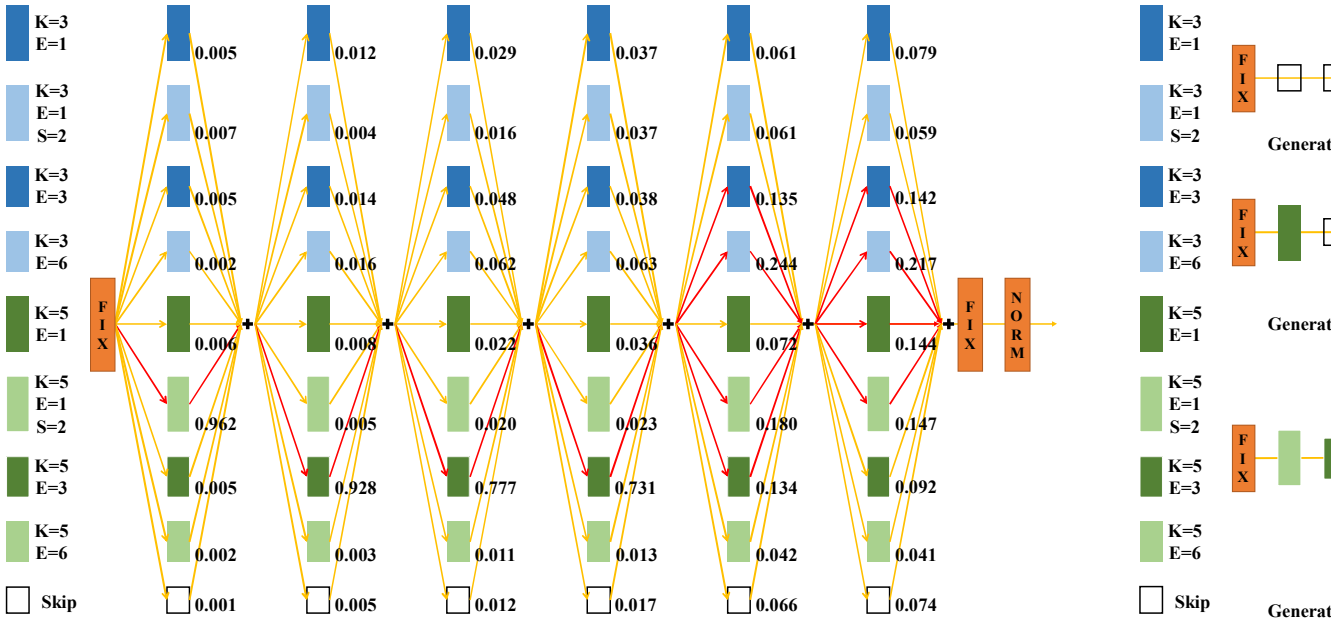


Figure 8: The optimized neural network generated by the NAS algorithm on the 100% performance hardware platform. The weights $m_{l,i}$ on each edge of the blocks are also marked, where the edges to selected blocks are rendered in red.

4. Experimental Results

In this section, we first introduce the benchmark datasets, experimental setup, and the testing platforms, then report results. The following experiments were conducted.

1. To evaluate the efficiency of our proposed pipeline, comparisons on the #parameters, #flops, latency, and accuracy are performed on HPatches [48] datasets with LF-Net [18], RF-Net [19], SIFT [9], SURF [10], ORB [26].
2. To evaluate the keypoint detection accuracy (repeatability), on HPatches [48] and EF [49] datasets, we compare our network with SIFT [9], FAST [50], ORB [26], SURF [10], RF-Det [19] and D2-Det [20].
3. To evaluate the accuracy of feature matching, we compare these methods' results of $1-10$ pixel matching thresholds in different illuminations and viewpoints on HPatches [48] datasets.
4. We also compare the performance of our entire 3D reconstruction system with ORB-SLAM2 [26], Elastic Fusion [42], RGBD TAM [43], and GCN SLAM [44] on the TUM [51] dataset. Note that since our focus is on the matching part, not the entire reconstruction pipeline, and we did not implement sophisticated bundle adjustment or global refinement, we only compared with these methods.

4.1. Dataset and Experimental Setup

Searching, Training, and Test Data. We used Photo Tourism [52] as searching data to perform NAS training. These data are from Photo Tourism reconstructions from Trevi Fountain (Rome), Notre Dame (Paris) and Half Dome (Yosemite). Each data sample consists of a series of corresponding patches, which are obtained by projecting 3D points from Photo Tourism reconstructions back into the original images. We used these data to create $500,000$ pairs of image patches and resized them to 32×32. We used the HPatches [48] dataset to train the feature description network. The dataset contains 116 sequences, each of which contains six pictures and their corresponding homographies. We used 90% of the sequence of view group as the training set, and the rest as the test set. We also included the EF [49] dataset into the test set.

Neural Architecture Search Process. To perform the NAS search, the latency table should be first built by estimating individual delay of each canonical candidate block on the deployment platform. Then the SuperNet can be trained more efficiently by using latency information from this table. We empirically set the training epoch number to 60. In each epoch, $100K$ random pairs out of the $500K$ patch pairs generated from the searching dataset were used. The training process first trains the operation weights w of the SuperNet, then optimizes the network weight η. The learning rate of w is set to 0.01 using SGD with momentum and the learning rate of η is set to 0.1 with Adam optimizer. The input patch size is set to 32×32, and the batch size is 256. Weights are trained on 80% of searching data, and η is trained on the remaining 20% of searching data. In order to control Gumbel softmax (Equation. 5), we use an exponentially decreasing temperature τ. Similar to FBNet [24], we used an initial temperature τ of 5.0 and exponentially anneal it by $exp(-0.045) \approx 0.956$ every epoch. When the search is done, we get the final network from the SuperNet based on the maximum probability blocks of each layer.

End-to-End Training Process. At training stage, we set the learning rate of descriptor and detector networks to 0.1. In each training batch, we first extract 512 keypoints from an image to train the keypoint detection network once, and then extract the corresponding 512 patches to train the feature description network twice. We resize all images to 640×480, and apply random rotations and translation transformations for data augmentation.

4.2. Feature Extraction with Networks Generated on Different Hardware Platforms

To test the hardware-adaptiveness of our proposed pipeline, we utilize a workstation with $E5\text{-}2690V3 \times 2$ CPU, NVIDIA GTX 2080×2 GPU, and 64G RAM. We downclocked the two GPUs to its original 50% and 25% clock frequencies, respectively. The

NAS is performed on these three different hardware platforms, using Photo Tourism datasets [52].

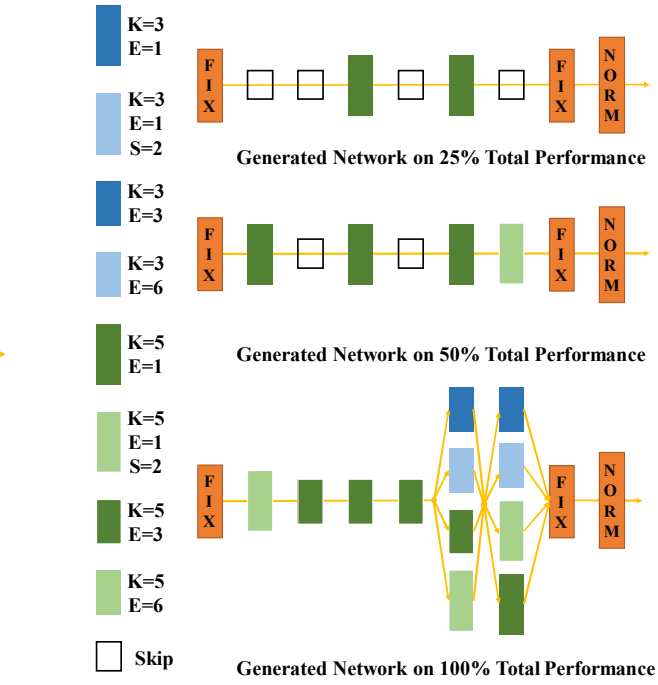


Figure 9: The generated results for three different hardware platforms.

The generated keypoint detector neural network for the 25% performance hardware has only one $k3\_e1$ block to transform the input image to the feature map. The 50% and 100%-performance platforms select two $k3\_e1$ blocks to process the input image to improve the accuracy of the extracted feature maps. All the three designs satisfy the latency requirement.

Figure 8 illustrates the optimized feature description neural network generated by the NAS algorithm on the 100%-performance hardware platform. The weights $m_{l,i}$ on each edge of the blocks are also marked, where the edges to selected blocks are rendered in red. When edge weights are larger than 0.1 the blocks are selected (e.g., in the last two cells). The final architectures for the feature description network on the three target hardware platforms are illustrated in Figure 9.

The neural network for 25%-performance workstation platform has the following structure: it utilizes two skip blocks to downsample the input image; then two $k5\_e1$ blocks separated by skip blocks to extract features. In the 50%-performance platform, two $k5\_e1$ blocks separated by skip blocks are selected first, followed by a $k5\_e1$ block and a $k5\_e1\_s2$ block. The 100%-performance hardware platform supports a more complicated network, organized by a $k5\_e1\_s2$ block and three $k5\_e3$ blocks first. Then two cells consisting of parallel blocks are selected. The first cell includes a $k3\_e3$ block, a $k3\_e6$ block, a $k5\_e3$ block, and a $k5\_e1\_s2$ block; the second cell includes a $k3\_e3$ block, a $k3\_e6$ block, a $k5\_e1\_s2$ block, and a $k5\_e1$ block. The parallel blocks improve the matching accuracy with only a limited decrease in processing efficiency on the 100%-performance platform.

4.3. Feature Extraction and Matching Results

We evaluate the feature description latency and accuracy of the generated neural network, and compare it with other classic geometry-based and learning-based algorithms. We consider a standard image matching scenario where features from two given images are extracted and matched. We use the sequences of full images provided by the HPatches dataset [48], and match one image with all other images in each sequence. The latency is measured by timing the feature extraction procedures. The accuracy is measured by the recall of the matching results, which denotes the number of correctly matched descriptors with respect to the number of corresponding descriptors between two images of the same scene. To evaluate the feature matching performance, we use the Mean Matching Accuracy (MMA) of three matching strategies [53].

1. Nearest Neighbor (NN): two descriptors A and B are matched if their distance is the nearest.
2. Nearest Neighbor with a threshold (NNT): two descriptors A and B are matched if their distance is the nearest and is below a threshold t.
3. Nearest Neighbor distance ratio (NNR): two descriptors A and B are matched if $\|D_A - D_B\|/\|D_A - D_C\| < t$ where D_B is the first, and D_C the second, nearest neighbor to D_A.

Note that all learned descriptors are $L2$ normalized. For fairness, we also normalize those manually designed feature descriptors and use the nearest neighbor distance ratio threshold $t = 0.8$. A 5-pixel threshold is used in the matching, and the results are shown in Table. 4.

The proposed algorithm generated three neural networks on the corresponding GPUs, achieving accuracy of 72% , 74%, and 78% respectively, all higher than the existing ones. For a fair comparison, we compared the speed of these three networks' architectures (and that of other existing approaches) on a same full-performance workstation. The network latency of these three architectures is 22.6ms, 31.1ms, and 38.7ms, respectively. Although the classic SURF algorithm has the smallest latency (10 ms), but it has significantly lower accuracy (43%) than these three architectures.

We compared the generated networks with the current state-of-the-art learning based methods and manually designed methods with a pixel thresholds from 1-10. All these experiments ran on the 100%-performance workstation. Figure 10 shows the MMA results of the overall, different illumination, and different viewpoint conditions, on the HPatchs dataset. As shown in the figure, our method achieves the best overall performances for 1-9 pixel thresholds. An illustration of the feature matching results of ORB [26], SIFT [9], RF-Net [19] and our method are shown in Figure 11.

We also tested and compared the repeatability of our generated keypoint detection network and other approaches. As shown in Table. 5, the keypoints detected by learning based end-to-end keypoint detection methods (LF-Det, RF-Net and ours) are more stable than FAST.

4.4. 3D Reconstruction Results

We evaluate the performance of our system in public datasets and real world environment to show the advances of our 3D reconstruction system. Note that the network was never trained on these reconstruction datasets. So the experiments also reveal the generalization of our system. Our 3D reconstruction system is coded using PyTorch's C++ version (libtorch) with CUDA support. We set the time spend on keypoints detection (1000 keypoints) to 15ms and feature description to 35ms (a $3:7$ predefined ratio), but the entire process including intermediate data transform and deploy, making the total time consumption to around 90ms. Together with the subsequent tracking, mapping

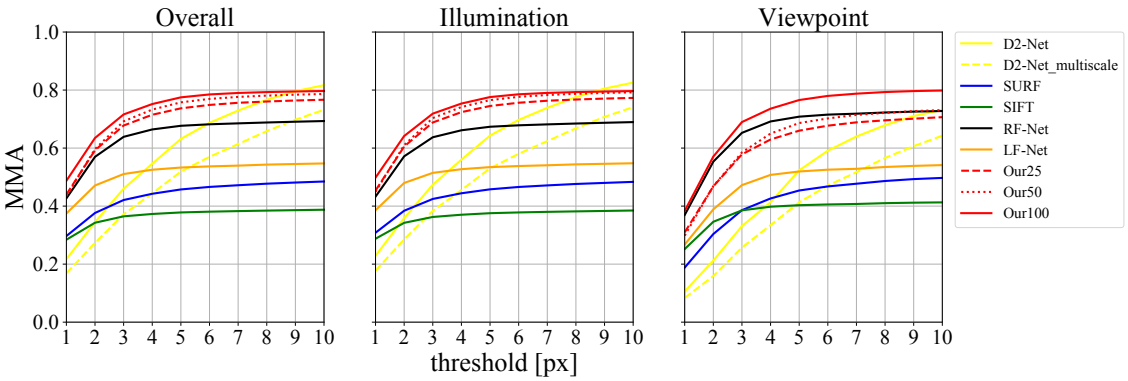


Figure 10: Evaluation on HPatches [48] Dataset. We use the mean matching accuracy (MMA) as a evaluation function for all methods. The methods used in this evaluation include D2-Net [20], SURF [10], SIFT [9], RF-Net [19], LF-Net [18]. Our method get the best performance on 1-9 pixel thresholds, and D2-Net can achieve better results on larger thresholds.

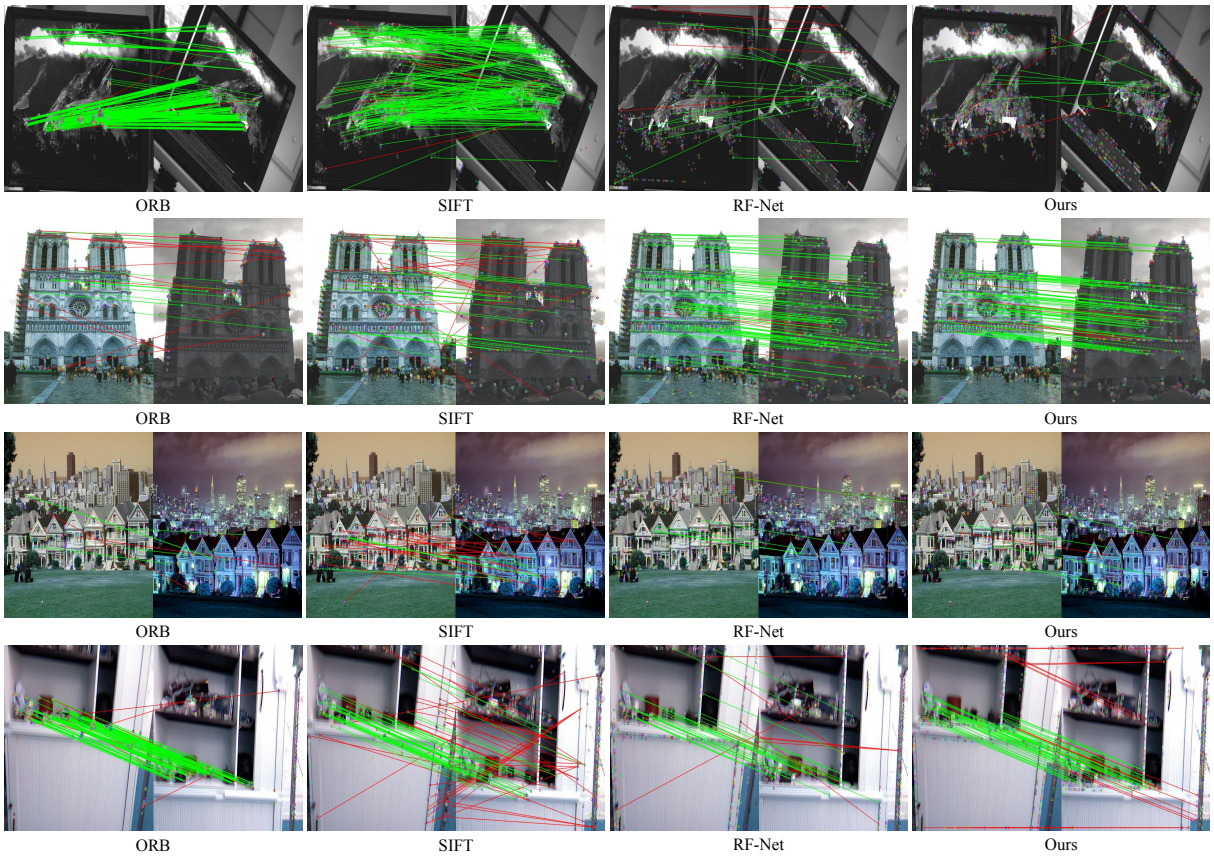


Figure 11: Feature matching results of ORB [26], SIFT [9], RF-Net [19] and our method. Correct matches are drawn with green lines and wrong matches with red lines. The top row images are from HPatches [48] Dataset, the middle two rows images are from EF [49] Dataset and the bottom row images are from TUM [51] fr1-room benchmark. We extract 1024 keypoints for each image and use the nearest neighbor distance ratio (0.8) as our matching strategy. We choose these images to show matching quality on different condition: viewpoint change, illumination change and small overlap. More green lines means better results.

and loop closure in parallel, our system runs at a speed of 10 to 15fps in all hardware platforms.

Table. 3 shows the camera pose estimation results of our proposed 3D reconstruction system (on the three hardware platforms) and original ORBSLAM2 on the TUM dataset [51] using public benchmark. We can see that most of the results have higher accuracy than other methods. Moreover, the camera pose estimation results on the 100% performance hardware framework is better than the others. Figure 12 shows the camera trajectory of our camera pose estimation results running on *TUM fr1_room* sequence.

We also tested our 3D reconstruction system by scanning an indoor scene. The capturing sensor we used is an Intel RealSense D415, a binocular depth camera mounted on a laptop, as shown in Figure 13. First, we ran 3D reconstruction on a laptop using the images captured by the RealSense sensor. Meanwhile, for demonstration and comparison purpose, we also stored the video and send it to the workstation, where 25% and 100% of its computation capability were used to test the corresponding learning pipelines. All these systems achieved real-time computing. In Figure 14, we show the reconstruction results computed using the 25% and 100% workstation pipelines. The laptop computa-

Table 3: Comparison of camera pose estimation accuracy between our proposed 3D reconstruction (on all three hardware platforms) with other camera pose estimation systems on TUM [51] dataset (in meters). The absolute trajectory error is used to measure the difference between points of the true and the estimated trajectory. The data are from [44].

Dataset	ORB SLAM2 [2]	Elastic Fusion [42]	RGBD TAM [43]	GCN SLAM [44]	Our 25	Our 50	Our 100
fr1_room	0.036	-	-	**0.021**	0.091	0.075	0.065
fr1_360	0.213	0.108	0.101	0.155	0.129	0.099	**0.088**
fr3_long_office	0.010	0.017	0.027	0.021	0.037	0.01	**0.009**
fr3_large_cabinet	-	0.099	**0.070**	**0.070**	0.108	0.101	0.093
fr3_nnf	-	-	-	0.086	0.0956	**0.043**	**0.043**

Table 4: Descriptor learning and matching result comparison on HPatches datasets [48]. The accuracy is calculated through mean matching accuracy (MMA). All results are obtained by running on the 100% performance workstation.

Algorithms	#Params	#FLOPs	Latency(ms)	Accuracy
SIFT [9]	-	-	55.9	0.38
SURF [10]	-	-	10.0	0.43
ORB [26]	-	-	49.3	0.29
LF-Net [18]	2642613	24.63 GB	38.4	0.56
RF-Net [19]	1356450	45.94 GB	86.9	0.68
Ours (25%)	280711	2.81GB	22.6	0.72
Ours (50%)	325695	5.87GB	31.1	0.74
Ours (100%)	360079	11.39GB	38.7	0.78

Table 5: Repeatability of keypoints in three evaluation sequences. Repeatability is defined as the mean percentage of points simultaneously present in the corresponding image pair.

-	HPatches-illum	HPatches-view	EF Dataset
DoG [9]	0.617	0.626	0.493
FAST [50]	0.769	0.681	0.533
ORB [26]	0.734	0.692	0.436
SURF [10]	0.679	0.625	0.536
RF-Det [19]	0.774	0.670	0.549
D2-Det [20]	0.620	0.393	0.445
Our (25%)	0.687	0.578	0.463
Our (50%)	0.704	0.584	0.458
Our (100%)	0.695	0.587	0.464

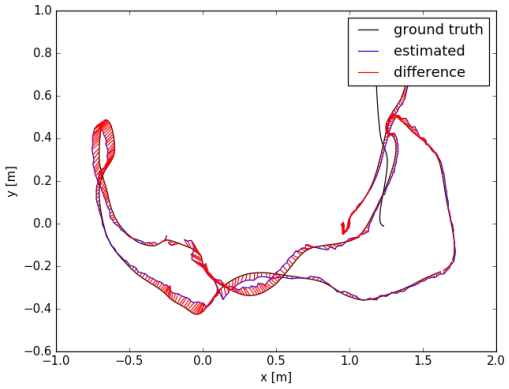


Figure 12: A camera pose estimation result of fr1_room sequence in TUM [51] dataset.

Figure 13: Laptop with Intel RealSense D415

tion result is very similar to the 100% workstation result. We can see that the results obtained by 100% performance hardware framework are more accurate: the corridor is straighter, and the room reconstruction is clearer.

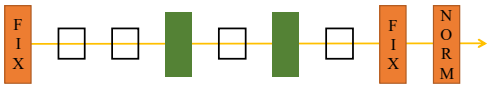
5. Conclusions

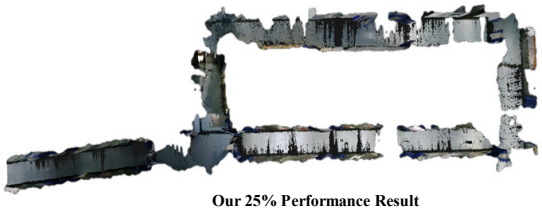
We proposed an automatic end-to-end feature extraction framework based on neural architecture search for real-time 3D reconstruction. This is a first framework to automatically generate and optimize learning-based feature extraction pipeline for real-time image matching and 3D reconstruction. We conducted various experiments on benchmarks and real-scenes to validate the efficiency and accuracy of the generated neural networks in feature modeling and matching. Our experimental results demonstrated that the proposed framework is flexible and adaptive to hardware, can achieve state-of-the-art performance on matching accuracy and efficiency, and can be used for real-time 3D reconstruction.

In our current experiment design, different hardware platforms are simulated by downclocking the GPU clock frequency different levels. However, besides clock speed, other factors such as kernel size and architectural difference could also affect the hardware performance. In the future, we will incorporate more factors into consideration to produce more accurate simulations and performance evaluation.

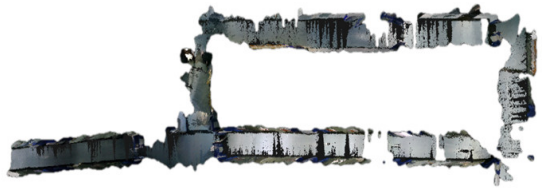
Acknowledgments

The work reported in this paper is supported by the National Key R&D Program of China 2018YFB1701300, National Natural Science Foundation of China 51805411, and National Science Foundation of USA OIA-1946231.





Our 25% Performance Result



Our 100% Performance Result

Figure 14: The 3D reconstruction results of an indoor scene using 25% and 100% performance hardware frameworks. Both systems obtain real-time computing.

References

[1] B. Fan, Q. Kong, X. Wang, Z. Wang, S. Xiang, C. Pan, P. Fua, A performance evaluation of local features for image-based 3d reconstruction, IEEE Transactions on Image Processing 28 (10) (2019) 4774–4789.

[2] R. Mur-Artal, J. D. Tardós, Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras, IEEE Transactions on Robotics 33 (5) (2017) 1255–1262.

[3] J. Engel, V. Koltun, D. Cremers, Direct sparse odometry, IEEE transactions on pattern analysis and machine intelligence 40 (3) (2017) 611–625.

[4] G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in: Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007, pp. 1–10.

[5] X. Li, S. Iyengar, On computing mapping of 3D objects: A survey, ACM Computing Surveys 47 (2) (2015) 34:1–34:45.

[6] A. Kim, R. M. Eustice, Real-time visual slam for autonomous underwater hull inspection using visual saliency, IEEE Transactions on Robotics 29 (3) (2013) 719–733.

[7] S. Zheng, J. Hong, K. Zhang, B. Li, X. Li, A multi-frame graph matching algorithm for low-bandwidth rgb-d slam, Computer-Aided Design 78 (2016) 107–117.

[8] C. Le, X. Li, Sparse3d: A new global model for matching sparse rgb-d dataset with small inter-frame overlap, Computer-Aided Design 102 (2018) 33–43.

[9] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International journal of computer vision 60 (2) (2004) 91–110.

[10] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-up robust features (surf), Computer vision and image understanding 110 (3) (2008) 346–359.

[11] M. Trajković, M. Hedley, Fast corner detection, Image and vision computing 16 (2) (1998) 75–87.

[12] N. Savinov, A. Seki, L. Ladicky, T. Sattler, M. Pollefeys, Quad-networks: unsupervised learning to rank for interest point detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1822–1830.

[13] K. Lenc, A. Vedaldi, Learning covariant feature detectors, in: European Conference on Computer Vision, Springer, 2016, pp. 100–117.

[14] D. Mishkin, F. Radenovic, J. Matas, Repeatability is not enough: Learning affine regions via discriminability, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 284–300.

[15] S. Zagoruyko, N. Komodakis, Learning to compare image patches via convolutional neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 4353–4361.

[16] K. M. Yi, E. Trulls, V. Lepetit, P. Fua, Lift: Learned invariant feature transform, in: European Conference on Computer Vision, Springer, 2016, pp. 467–483.

[17] D. DeTone, T. Malisiewicz, A. Rabinovich, Superpoint: Self-supervised interest point detection and description, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2018, pp. 224–236.

[18] Y. Ono, E. Trulls, P. Fua, K. M. Yi, Lf-net: learning local features from images, in: Advances in Neural Information Processing Systems, 2018, pp. 6234–6244.

[19] X. Shen, C. Wang, X. Li, Z. Yu, J. Li, C. Wen, M. Cheng, Z. He, Rf-net: An end-to-end image matching network based on receptive field, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 8132–8140.

[20] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, T. Sattler, D2-net: A trainable cnn for joint description and detection of local features, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 8092–8101.

[21] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.

[22] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, arXiv preprint arXiv:1806.09055 (2018).

[23] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le, Mnasnet: Platform-aware neural architecture search for mobile, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[24] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10734–10742.

[25] E. Rosten, R. Porter, T. Drummond, Faster and better: A machine learning approach to corner detection, IEEE transactions on pattern analysis and machine intelligence 32 (1) (2008) 105–119.

[26] E. Rublee, V. Rabaud, K. Konolige, G. R. Bradski, Orb: An efficient alternative to sift or surf., in: ICCV, Vol. 11, Citeseer, 2011, p. 2.

[27] Y. Verdie, K. Yi, P. Fua, V. Lepetit, Tilde: a temporally invariant learned detector, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5279–5288.

[28] L. Zhang, S. Rusinkiewicz, Learning to detect features in texture images, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6325–6333.

[29] K. Moo Yi, Y. Verdie, P. Fua, V. Lepetit, Learning to assign orientations to feature points, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 107–116.

[30] L. Wang, Y. Qiao, X. Tang, Action recognition with trajectory-pooled deep-convolutional descriptors, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 4305–4314.

[31] X. Han, T. Leung, Y. Jia, R. Sukthankar, A. C. Berg, Matchnet: Unifying feature and metric learning for patch-based matching, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3279–3286.

[32] C. B. Choy, J. Gwak, S. Savarese, M. Chandraker, Universal correspondence network, in: Advances in Neural Information Processing Systems, 2016, pp. 2414–2422.

[33] V. Balntas, E. Riba, D. Ponsa, K. Mikolajczyk, Learning local feature descriptors with triplets and shallow convolutional neural networks., in: Bmvc, Vol. 1, 2016, p. 3.

[34] V. Kumar BG, G. Carneiro, I. Reid, Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 5385–5394.

[35] Y. Tian, B. Fan, F. Wu, L2-net: Deep learning of discriminative patch descriptor in euclidean space, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 661–669.

[36] A. Mishchuk, D. Mishkin, F. Radenovic, J. Matas, Working hard to know your neighbor's margins: Local descriptor learning loss, in: Advances in Neural Information Processing Systems, 2017, pp. 4826–4837.

[37] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning, Springer, 2019.

[38] X. Dong, Y. Yang, Searching for a robust neural architecture in four gpu hours, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 1761–1770.

[39] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: Bridging the depth gap between search and evaluation, arXiv preprint arXiv:1904.12760 (2019).

[40] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, H. Xiong, Pc-darts: Partial channel connections for memory-efficient differentiable architecture search, arXiv preprint arXiv:1907.05737 (2019).

[41] M. R. U. Saputra, A. Markham, N. Trigoni, Visual slam and structure from motion in dynamic environments: A survey, ACM Computing Surveys (CSUR) 51 (2) (2018) 1–36.

[42] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, A. Davison, Elasticfusion: Dense slam without a pose graph, Robotics: Science and Systems, 2015.
[43] A. Concha, J. Civera, Rgbdtam: A cost-effective and accurate rgb-d tracking and mapping system, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 6756–6763.
[44] J. Tang, L. Ericson, J. Folkesson, P. Jensfelt, Gcnv2: Efficient correspondence prediction for real-time slam, IEEE Robotics and Automation Letters 4 (4) (2019) 3505–3512.
[45] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.
[46] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.
[47] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[48] V. Balntas, K. Lenc, A. Vedaldi, K. Mikolajczyk, Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5173–5182.
[49] C. L. Zitnick, K. Ramnath, Edge foci interest points, in: 2011 International Conference on Computer Vision, IEEE, 2011, pp. 359–366.
[50] E. Mair, G. D. Hager, D. Burschka, M. Suppa, G. Hirzinger, Adaptive and generic corner detection based on the accelerated segment test, in: European conference on Computer vision, Springer, 2010, pp. 183–196.
[51] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of rgb-d slam systems, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.
[52] N. Snavely, S. M. Seitz, R. Szeliski, Photo tourism: exploring photo collections in 3d, in: ACM Siggraph 2006 Papers, 2006, pp. 835–846.
[53] K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors, IEEE transactions on pattern analysis and machine intelligence 27 (10) (2005) 1615–1630.