FastStereoNet: A Fast Neural Architecture Search for Improving the Inference of Disparity Estimation on Resource-Limited Platforms

Mohammad Loni, *Student Member, IEEE,* Ali Zoljodi, Amin Majd, *Member, IEEE,* Byung Hoon Ahn, *Student Member, IEEE,* Masoud Daneshtalab, *Senior Member, IEEE,* Mikael Sjödin, and Hadi Esmaeilzadeh, *Member, IEEE*

***Abstract*—Convolutional Neural Networks (CNNs) provide the best accuracy for disparity estimation. However, CNNs are computationally expensive, making them unfavorable for resource-limited devices with real-time constraints. Recent advances in Neural Architectures Search (NAS) promise opportunities in automated optimization for disparity estimation [1], [2]. However, the main challenge of the NAS methods is the significant amount of computing time to explore a vast search space (e.g., 1.6×10^{29} [3]) and costly training candidates. To reduce the NAS computational demand, many proxy-based NAS methods have been proposed. Despite their success, most of them are designed for comparatively small-scale learning tasks. In this paper, we propose a fast NAS method, called FastStereoNet, to enable resource-aware NAS within an intractably large search space. FastStereoNet automatically searches for hardware-friendly CNN architectures based on Late Acceptance Hill Climbing (LAHC), followed by Simulated Annealing (SA). FastStereoNet also employs a fine-tuning with transferred weights mechanism to improve the convergence of the search process. Collection of these ideas provides competitive results in terms of search time and strikes a balance between accuracy and efficiency. Compared to the state-of-the-art [1], FastStereoNet provides 5.25× reduction in search time and 44.4× reduction in model size. This benefits are attained while yielding a comparable accuracy that enables seamless deployment of disparity estimation on resource-limited devices. Finally, FastStereoNet significantly improves the perception quality of disparity estimation deployed on FPGA and Intel® NCS2 accelerator in a significantly less onerous manner.**

***Index Terms*—Machine Vision, Disparity Estimation, Optimization, Neural Architecture Search, Transfer Learning**

I. Introduction

DISPARITY estimation is the problem of finding corresponding pixels in multiple images of a scene from different viewpoints. Disparity estimation is a key task in the processing pipeline of extracting depth information from stereo images. Disparity estimation is widely used in many applications, such as remote handling with depth models [4], image segmentation [5], and 3D-modelling of natural objects [6]. CNNs provide highly accurate depth estimation results [7], [8], [9], [10]. However, the massive computational intensity of CNNs requires energy-hungry GPUs to provide real-time performance for disparity estimation [11]. Concurrently, there is an increasing demand for deploying CNNs onto resource-limited devices due to connectivity, privacy, and efficiency concerns [12], [13].

In cyber-physical systems, as embedded vessels of computation, Field Programmable Gate Arrays (FPGAs) represent a power efficient alternative to GPUs while offering versatility over Application-Specific Integrated Circuits (ASICs). However, FPGAs in embedded systems are often resource constrained to meet the power envelop. This poses a significant challenge for the FPGA implementation of CNN-based disparity estimation, while they have traditionally been used to realize conventional methods of disparity estimation (Section II-A). However, the accuracy of the conventional methods is not satisfactory in comparison with CNN-based techniques [11], [14], [15], [16]. Intel® Neural Compute Stick 2 (NCS2) is another embedded accelerator providing rapid development, supporting low-power embedded applications, and operating without cloud compute dependence. However, limited processing resources (on-chip memory<500MB) prevent their usage in implementing CNN-based disparity estimation.

Neural Architecture Search (NAS) research has shown significant progress in enabling resource-efficient neural architectures [17], [18], [19], [20], [21]. NAS is the process of automatically optimizing a neural network architecture. However, a large majority of NAS methods suffer from significant computational time [22] due to *directly* searching an exponentially large design space. As such, some recent NAS methods customize the search space by stacking copies of pre-trained neural cells [1], [23], [22], where each cell is usually well-optimized for proxy tasks. Although this technique decreases the search time, it might not be optimal for new unseen tasks.

In this paper, we propose FastStereoNet, a fast-and-efficient but straightforward multi-stage optimization method that directly explores a large search space for the disparity estimation task. FastStereoNet utilizes an ordered sequence of Late Acceptance Hill Climbing (LAHC) and Simulated

Manuscript received ...; revised ...; accepted ... . Date of publication ...; date of current version ... . *(Corresponding author: Mohammad Loni.)*

Mohammad Loni, Ali Zoljodi, and Mikael Sjödin are with the School of Innovation, Design and Engineering, Mälardalen University, 72218, Västerås, Sweden (e-mail: {mohammad.loni, ali.zoljodi, mikael.sjodin}@mdh.se).

Masoud Daneshtalab is with the School of Innovation, Design and Engineering, Mälardalen University, 72218, Västerås, Sweden (e-mail: masoud.daneshtalab@mdh.se) and the Department of Computer Systems, TalTech University, Ehitajate tee 5, 19086, Tallinn, Estonia.

Amin Majd is with the Department of Economics and Business Analysis, Arcada University of Applied Sciences, Jan-Magnus Janssonin aukio 1, 00560 Helsinki, Finland (e-mail: amin.majd@arcada.fi).

Byung Hoon Ahn and Hadi Esmaeilzadeh are with **A**lternative **C**omputing **T**echnologies (**ACT**) Lab, Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093-0404 USA (e-mail: {bhahn,hadi}@eng.ucsd.edu).

Annealing (SA) as the optimization stages. The reason why the proposed search method is substantially fast comes from the single solution based nature of SA, while for example, the genetic algorithm is relatively slow due to their multi-sample population-based evolutionary strategy (Appendix B). The convergence of the SA algorithm is highly sensitive to its initial candidate that is usually selected randomly. Therefore, we employed LAHC as the first stage of the optimization to provide initial candidates for SA and as such speedup the SA with a better initial condition. FastStereoNet considers *accuracy* and *estimated inference time* as the search objectives to design resource-efficient neural architectures with acceptable accuracy. Importantly, FastStereoNet uses a latency predictor to improve the overall speed of the neural architecture search. The experimental results (Appendix G) show that the R^2 score of the FastStereoNet latency predictor for GPU, FPGA, and Intel® NCS2 is 0.998, 0.98, and 0.97, respectively. We also considered the number of floating-point operations (FLOPs) to verify the search efficiency of the latency predictor. The correlation between FLOPs and measured GPU latency for the disparity estimation task is 0.68.

FastStereoNet leverages a discrete design space, represented as a Directed Acyclic Graph (DAG), while its vertices are the computational blocks of the CNN and the edges are the skip connections (Section III). The utilized discrete design space provides diverse architectures competitive with cell-based neural architecture design spaces. Inspired by [24], we use Siamese neural architecture as the backbone of the disparity estimation. FastStereoNet also utilizes a fine-tuning module for weight transfer across networks to improve the accuracy of candidates during the search steps (Section III-C). This paper makes the following contributions.

- We introduce a multi-objective Neural Architecture Search algorithm that combines Late Acceptance Hill Climbing followed by Simulated Annealing.
- We devise a latency predictor to accurately estimate the inference time of the candidate neural architectures on a range of target devices.
- We integrate a transferred weights mechanism that reuses the weights while training the candidate neural architectures to expedite the overall search.
- We extend the original LAHC and SA algorithms to guarantee a deterministic termination condition.

According to our experiments on KITTI 2015 dataset [25], we achieved 2× reduction in FPGA latency while yielding higher accuracy compared to state-of-the-art. Furthermore, the design time to find the CNN architecture is only 8 GPU days compared to 42 GPU days. This is, FastStereoNet requires 5.25× less search time to find the most accurate architecture for disparity estimation reported by AutoDispNet [1]. Out of all leading computation devices, Intel® NCS2 device is an emerging platform for accelerating CNNs on resource-limited embedded systems. To the best of our knowledge, FastStereoNet is the first method making the real-time implementation of accurate CNN-based disparity estimation on Intel® NCS2 [26] possible (Section VI-A) by reducing the size of memory footprint and using primitive CNN operations.

II. Related Work

To the best of our knowledge, FastStereoNet is the fist global search (macro NAS) method that yields competitive results compared to cell-based methods (micro NAS) in terms of search cost while meeting the resource constraints of the target embedded devices. To this end, we first present the potential resource-limited devices for accelerating disparity estimation. Then, we review the recent research advances in Neural Architecture Search (NAS).

A. Disparity Estimation on Resource-limited devices

Field Programmable Gate Array (FPGA). FPGAs are popular platforms for embedded devices that are successfully adopted for conventional disparity estimation methods with real-time constraints [14], [11], [15], [27]. Table VII compares the performance of different triangulation-based methods implemented on FPGA (Section VI-G). Although the inference time of the FastStereoNet on FPGA is still less than some of the triangulation-based methods, FastStereoNet is the first attempt that remarkably reduces the existing performance gap and delivering a higher accuracy by optimizing CNN architecture.

Intel® NCS2 accelerator. Intel® NCS2 is another embedded accelerator providing rapid development, supporting low-power embedded applications, and operating without cloud compute dependence. To the best of our knowledge, FastStereoNet is the first successful implementation of accurate disparity estimation on Intel® NCS2 (Section VI-A).

B. Neural Architecture Search (NAS)

Inspired by [28], we first categorize NAS methods as *macro NAS* and *micro NAS*. We review the prior works that utilize proxy tasks to improve NAS efficiency since dealing with the time-consuming evaluation and huge search space are the main challenges in NAS [23], [29]. Since we utilize a discrete search space, we review substantial research studies in this area.

1) Macro NAS: Macro NAS methods try to *directly* design the entire neural network architecture from scratch [30], [31], [32], [21], [33]. In other words, NAS finds an optimal architecture within a huge search space with the granularity of operations. It provides a high flexibility search space. However, the larger search space come at the cost of higher search cost.

Reinforcement learning (RL) is a popular approach for generating neural architectures [32], [21], [3], [30]. [32] proposed an LSTM-based controller to configure the CNN descriptions, then trains this LSTM with RL to maximize the classification accuracy. [3] proposes ENAS, a fast and efficient NAS trying to search an optimal sub-graph within a large computational graph by employing a meta-controller. In macro NAS, the size of search space exponentially grows with the increasing depth of a network [30], [32], e.g., a network with less than 12 possible result in 1.6×10^{29} distinct architectures [3]. It is not feasible to efficiently search such a large space in a reasonable amount of time (requiring thousand GPU hours). [3], [30], [32] prune the search space by limiting the depth of CNNs leading

to lower accuracy [28]. In contrast, FastStereoNet can directly search a vast search space in a reasonable amount of time.

Alternatively, a group of research studies rely on the evolutionary search methods where the best architecture is designed by iteratively refining a population of candidate architectures [34], [35], [36], [23], [37], [38]. However, these methods are costly, requiring hundreds of GPU days. FastStereoNet cuts down the search time to find optimal solutions despite large design spaces. DenseDisp [2] is a simulated annealing-based search method. Compared to DenseDisp, FastStereoNet provides more accurate results by utilizing a more complex space and a latency prediction model (Section VI-A).

2) Micro NAS: Micro NAS methods search the inner architecture of learning cells, while the interconnection among neural cells is defined by stacking several copies of the discovered cells [23], [3], [39], [29], [1], [28]. Although *micro NAS* methods decrease the search time, they might not be optimal for any unseen tasks since each cell is targeted to proxy tasks.

NASNet [29] is a popular micro NAS that searches neural cells on a small dataset, then transfer the discovered cells to a larger dataset. To search neural cells, Real et al. [23] proposed an evolutionary algorithm with modified tournament selection. [28] proposes GDAS, a gradient-based search using a learnable differentiable architecture sampler to avoid considering all the possible architectures in search space. Although the micro NAS employs an efficient search space to design the entire network, they are mainly optimized for comparatively small-scale datasets, e.g., CIFAR-10 dataset [29], [3], which do not guarantee optimal performance with large-scale datasets [1], [21]. Saikia et al. [1] proposes AutoDispNet, an efficient NAS for large-scale encoder-decoder architecture. FastStereoNet provides a more efficient search strategy because it is roughly 5.25× faster than AutoDispNet while searching a larger space. Also, FastStereoNet provides better latency-error trade-off by providing approximately 45.5× higher Network Information Density ($NID = \frac{Accuracy}{NetworkParameters}$). All in all, FastStereoNet maximizes design flexibility by enabling search at the granularity of primitive operations by adopting insights and methods from macro NAS.

3) Improving NAS Efficiency: To improve NAS speed and to reduce the NAS computational cost, a variety of techniques have been proposed to utilize proxy tasks [40]. HyperNet generates weights for candidate networks and evaluates them without full training from scratch [31]. [17], [18], [41] use partial training for accuracy prediction at the cost of noisy evaluations. Sharing weights among potential networks decreases the search time by two orders of magnitude [3]. [30], [42], [19], [43] use network to network transformation for reusing weights of previously discovered networks to amortize the training cost. In this work, we employed a transferred weights mechanism (Section III-C) to expedite the search (Section VI-D).

III. Architecture Design Space

In this section, we first present the overview of a Siamese neural architecture. Then, we describe the genotype encoding that represents the architecture of each candidate.

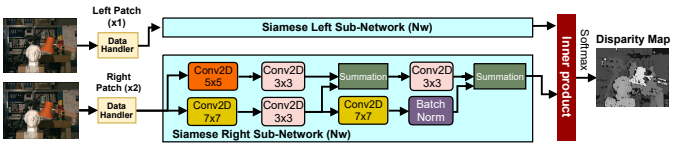


Fig. 1: The overview of the Siamese network architecture.

A. Siamese Network Architecture

In general, Siamese neural network is a learning model that tries to learn the similarity of its input images, instead of direct classification. Siamese neural networks contain two (or more) identical sub-networks, $N_W(X)$ parameterized by the same weight W, to extract the similar features between two input data (X_1, X_2). The objective of the training for Siamese networks is to find W such that the the Euclidean distance $d(X_1, X_2) = |N_W(X_1) - N_W(X_2)|$ is minimized for pair of similar images. We used a flexible Siamese neural network as the backbone architecture for disparity estimation, where each sub-network processes the left or right image (Fig. 1). While the use of the Siamese network in this paper is similar to [24], there is a significant difference. Instead of using hand-crafted design, we leveraged neural architecture search to automatically find an efficient sub-network ($N_W(X)$). At the end of the Siamese network, we placed an light-weight inner-product layer to join $N_W(X_1)$ and $N_W(X_2)$ leading to faster inference speed.

Training. In this paper, the image pairs are assumed to be rectified. Therefore the horizontal image axis is aligned with the epipolar lines. To train the network, [24] uses small patches sequentially extracted at random from the dataset for which ground-truth is available (managed by data handler). This strategy improves the diversity of examples and provides higher memory efficiency. While the size of the left image patch is equal to the network's receptive field, a larger patch is used for the right image. The features extracted by the aforementioned sub-networks are fed into the inner-product layer to compute the score of each disparity range (typically 128 or 256). Assuming (x_i, y_i) is the image coordinates of the center of the patch extracted from the left image, and d_{x_i,y_i} is the corresponding ground-truth disparity, we minimize the cross-entropy loss:

$$\min_{W} \sum_{i,y_i} p_{gt}(y_i) \log p_i(y_i, W) \tag{1}$$

We are interested in the 3-pixel error metric as the default disparity comparison measurement. Thus, we utilize a smooth target distribution $p_{gt}(y_i)$ that is centered around the ground-truth $(y_i)^{GT}$. For example,

$$p_{gt}(yi) = \begin{cases} \lambda_1 & if y_i = {y_i}^{GT} \\ \lambda_2 & if |y_i - {y_i}^{GT}| = 1 \\ \lambda_3 & if |y_i - {y_i}^{GT}| = 2 \\ 0 & otherwise \end{cases} \tag{2}$$

where we select $\lambda_1 = 0.5$, $\lambda_2 = 0.2$, and $\lambda_3 = 0.05$. The aim is to minimize cross-entropy for classification.

B. Representation of CNN Architectures

We represent the design space by using a single Directed Acyclic Graphs (DAG). Fig. 2 shows an example of a DAG, where a CNN architecture is a genotype directly encoded by two concurrent Lists, *List A* and *List B*, containing computational nodes [18]. The advantage of this representation is its flexibility, supporting variable-length architectures with arbitrary skip connections.

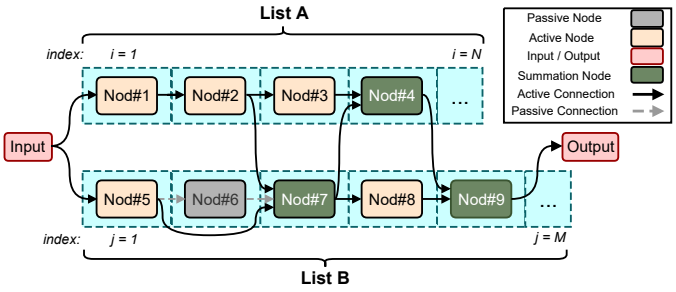


Fig. 2: A generic example of a genotype representing a CNN architecture in the design space.

Let us assume *List A* and *List B* have N and M columns, respectively. In this representation, the genotype is a flexible-length representation, where the size of N and M varies during network evolution. Inspired by [18], we define twenty-six types of node operations including summation, batch normalization, and variegated combinations of 2D convolution (Conv2d) settings as summarized in Table I. In this paper, we do not use the Pooling layer since the size of feature maps in each dimension should be odd. Importantly, we only used the primitive operations to support a wider range of devices, e.g., Intel® NCS2 does not execute the TensorFlow `RandomUniform` operation or separable convolution layer at the time of submitting the paper. The summation function adds two feature maps in an element-wise manner. If the input feature maps of summation node have different sizes of width and height, we resize the bigger image to match the small image by using `image.resize_with_crop_or_pad` function in TensorFlow [44]. To compensate the information loss from resizing, we apply a convolution layer connected to the batch normalization on the resized feature map before it is fed into the summation node. In the case of having a different number of channels, we pad the smaller feature maps with zeros. While adding the edges between the nodes, some nodes become isolated (*Nod#6* in Fig. 2). We consider them as *passive nodes* and ignore them during the training.

TABLE I: The specification of valid node operations supported by all commodity devices.

Node Operation	Parameter	Value
2D convolution (Conv2d)	padding	{Same, Valid}
	filter size	{32, 64}
	kernel size	$\{3\times3, 5\times5, 7\times7, 9\times9, 11\times11, 13\times13\}$
Summation		-
Batch Normalization		-

Network Construction Procedure. Here, we briefly describe the *LEGO-like stacking procedure*. ❶ First, we fill both *List A* and *List B* with random operations selected from Table I. The initial size of N and M is 10 in our experiments. ❷ Next, we connect the j^{th} node to the $(j-1)^{th}$ node in the same List. However, if j^{th} node in a list is a summation node, we connect the $(j-1)^{th}$ node of both lists as the summation requires two inputs (*Nod#7* in Fig. 2). The output feature map of the summation node will be connected to a copy of the same summation node in the other List (*Nod#4* in Fig. 2) and the next node in the same List. ❸ We connect the summation of output feature maps received from *List A* and *List B* to the output node (*Nod#9* in Fig. 2). FastStereoNet supports two types of mutation operations: swapping of two random nodes, and replacing a node operation with another valid operation.

The size of design space depends on the length of *List A* and *List B* in the genotype. The minimum size of the design space is 2×24^{10} since both *List A*, and *List B* consist of 10 nodes at the beginning of the search. However, the length of these two Lists may increase after mutation. For example, during the mutation, a mutated Conv2d with *Valid* padding must be replaced by two new Conv2d operations with *Valid* padding such that the old Conv2d kernel size is equal to the summation of the two new Conv2d kernel sizes minus one. We limit the maximum size of each list to 30. Therefore, the maximum size of the design space is 2×24^{30}.

C. Efficient NAS by Transferred Weights Mechanism

In each iteration, we evaluate each network candidate by fully training from scratch. Due to the enormous training cost (up to 2 GPU hours/network), iterative search approaches would be impractical. Therefore, to accelerate the evaluations, we propose a transferred weights mechanism that fine-tunes the training of the mutated architecture (child) by reusing the information stored on the previous architecture (parent). This mechanism allows the child architecture to inherit the knowledge from its parent and achieve higher accuracy with the same training epochs. Assume we mutate the j^{th} node

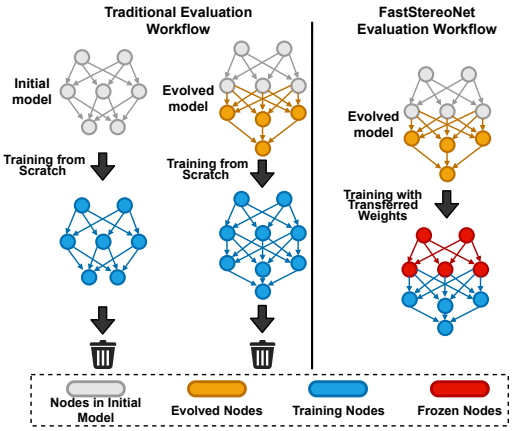


Fig. 3: Comparing the traditional training workflow and the fine-tuning with transferred weights mechanism. FastStereoNet reuses shared weights of the previous architecture (parent) with the new architecture (child) to improve evaluations.

of the *List A*. Weights of all the nodes preceding the j^{th} node in both Lists are reused from the previous architecture (parent) and are frozen in the new training procedure of new architecture (child). Fig. 3 compares the workflow of the full training evaluation (traditional) with the proposed weight-reusing training evaluation. Only the blue nodes in Fig. 3 will be trained after evolving network, leading to faster training. Experimental results (Section VI-C) show that fine-tuning with transferred weights mechanism provides a significantly faster convergence compared to the traditional methods.

IV. Search Strategy

Neural Architecture Search (NAS) is an NP-hard problem with an exponential time complexity [36]. In other words, there is no polynomial optimization method to find the optimal solution in a reasonable amount of time. Also, an exhaustive search for finding a global optimum is infeasible even for small design spaces, e.g., [17] requires 334 GPU days to exhaustively search a design space containing only 8000 solutions. To this end, we leverage a meta-heuristic search to deal with the exponential complexity of the NAS problem.

A. Proposed Method

Inspired by [45], [46], we propose a multi-stage search method with two major stages, comprising Late Acceptance Hill Climbing (LAHC) [47], and Simulated Annealing (SA) [48], [49], respectively. Algorithm 1 summarizes the pseudo-code of the FastStereoNet. The inputs of the FastStereoNet are the training and search configuration specified in Table III. The LAHC algorithm, as the semi-local search engine, quickly searches the space to bias the initial solution of SA. The key motivation of leveraging LAHC as the first search stage is to improve the SA convergence by providing better initial conditions. SA, as the global search engine, starts with a solution augmented by LAHC. LAHC mainly explore the search space, while SA tries to exploit the search space.

Compared to discrete search methods [36], [50], differentiable architecture search methods provide promising results by designing and training the network architecture at the same time [39]. However, differentiable architecture search methods are not ultimate solutions for any given task while discrete search methods provide more efficient results. For example, SI-EvoNet provides more accurate results with 1.84x higher compression rate and 2.1x faster search time compared to DARTS [39] search method on CIFAR-10 [50]. This is due the fact that 1) differentiable architecture search methods suffer from inefficient training [51]; 2) different differentiable architecture search methods converge to similar results [52] with equal search spaces and training setups, suggesting that differentiable architecture search methods are not capable of discovering diverse set of solutions. To this end, we propose a multi-stage iterative method as the optimization algorithm to provide higher discovery proficiency. In the rest of this section, we explain the LAHC and SA algorithms and how we tweak them for our problem. Finally, we analyze the time complexity and the proof of convergence of FastStereoNet.

Algorithm 1 Pseudo-code of the FastStereoNet Algorithm

```
function FastStereoNet():
    LocalOptimum ← LAHC (RandomSolution)
    GlobalOptimum ← SA (LocalOptimum)
    return GlobalOptimum

function LAHC(RandomSolution):
    i ← 1
    idle ← 0
    CurrentSolution ← RandomSolution
    while i ⩽(Steps×MinRate)∨((idle<IdleFraction×i)∧(i⩽Steps×MaxRate)) do
        NewSolution ← Mutate (CurrentSolution)
        E1 ← Energy (CurrentSolution)
        E2 ← Energy (NewSolution)
        if E2 < E1 then
            CurrentSolution ← NewSolution
        else
            idle += 1
        i += 1
    return CurrentSolution

function SA(LocalOptimum):
    T_Min , T_Max ← Scheduler (MaxRateT_Max, MaxRateT_Min, Steps)
    GlobalOptimum ← Annealer (LocalOptimum, T_Min, T_Max, Steps, MaxRateAnnealing)
    return GlobalOptimum

function Scheduler(MaxRateT_Max, MaxRateT_Min, Steps):
    T ← |Energy (RandomSolution_1) - Energy (RandomSolution_2)|
    i ← 0
    while AcceptanceRate > 0.98 ∧ i ⩽ MaxRateT_Max × Steps do
        T ← Round ((T / 1.5), 2)
        Run (Steps)
        i += 1
    i ← 0
    while AcceptanceRate < 0.98 ∧ i ⩽ MaxRateT_Max × Steps do
        T ← Round((T × 1.5), 2)
        Run (Steps)
        i += 1
    T_Max ← T
    i ← 0
    while ImprovementRate > 0 ∧ i ⩽ MaxRateT_Min × Steps) do
        T ← Round ((T / 1.5), 2)
        Run (Steps)
        i += 1
    T_Min ← T
    return T_Min, T_Max

function Annealer(LocalOptimum, T_Min, T_Max, Steps, MaxRateAnnealing):
    T_Factor ← - LOG (T_Max / T_Min)
    i ← 0
    Step_total ← Steps × MaxRateAnnealing
    CurrentSolution ← LocalOptimum
    while i ⩽ Step_total do
        i += 1
        NewSolution ← Mutate (CurrentSolution)
        E1 ← Energy (CurrentSolution)
        E2 ← Energy (NewSolution)
        if E2 < E1 then
            CurrentSolution ← NewSolution
        else
            Random (r∈(0, 1))
            Δ ← E2 - E1
            if r < EXP(-Δ / T) then
                CurrentSolution ← NewSolution
        T ← T_Max × EXP (T_Factor × (i/Step_total))
    return CurrentSolution
```

Stage 1: Late Acceptance Hill Climbing (LAHC). LAHC is a heuristic search method that is responsible for providing initial solutions for the SA algorithm. LAHC is an extension of the simple hill climbing algorithm [53], while worse solutions are allowed to be accepted in LAHC in hopes of finding a better solution in future [47]. Function LAHC in Algorithm 1 describes the pseudo-code of the LAHC.

The LAHC tries to delay getting stuck in a local optimum by continuing the search procedures until the total number solutions with no improvement exceeds a descending threshold, named `IdleFractionRatio`, which is equal to `IdleFraction`×`i`. `IdleFraction` is a constant value, and `i` is the loop counter. The original LAHC algorithm only considers the `IdleFractionRatio` as the search termination condition. Considering the huge size of FastStereoNet search space and costly evaluating each candidate, the original LAHC algorithm is not a fast search method in practice since `IdleFractionRatio` is not a deterministic upper-bound termination condition. Therefore, we extend the original LAHC algorithm by limiting the maximum number of search iterations. The new search termination condition is defined in Equation 3, where `TotalIdleSolutions` is the accumulated number of unsuccessful solutions after the last improvement, and `MinimumSteps` and `MaximumSteps` are lower-bound and upper-bound for the search process, respectively. While the proposed extended LAHC is fast, it cannot be used in isolation because its performance is not still competitive with more complex meta-heuristics (Section VI-C). To this end, we complement this by augmenting SA to the search.

$$\begin{aligned} \forall i \in 1..N \Rightarrow & ((\texttt{TotalIdleSolutions} < \texttt{IdleFractionRatio}) \\ & and\ (i \leqslant \texttt{MaximumSteps})) \\ & or\ (i \leqslant \texttt{MinimumSteps}) \end{aligned} \tag{3}$$

Stage 2: Simulated Annealing (SA). SA, on the other hand, is a guided optimization method in charge of finding the global optimum in the design space. SA avoids getting trapped in a local optimum by accepting worse neighbors with a varying probability. As such, SA enables the fast convergence to global optimum solution. The proof for the convergence of SA is presented in Section IV-B. Function SA in Algorithm 1 describes the pseudo-code of the SA. The ENERGY function in the algorithm is the search objective function. SA first begins by producing the new solutions by using the MUTATION function described in Secrtion III-B. Then it accepts the new solution if it has lower ENERGY function (objective function), while a worse solution is accepted with the probability of MIN(1, EXP(-Δ / `T`). Δ is the difference of search objectives of the current solution and the new generated solution (Δ = ENERGY (`NewSolution`) - ENERGY (`CurrentSolution`)), and `T` is the controlling parameter which corresponds to the temperature of the annealing procedure. `T` is gradually decreased according to the predefined minimum and maximum temperatures ($\texttt{T}_{\texttt{Max}}$ and $\texttt{T}_{\texttt{Min}}$). SA starts with a high value of `T` ($\texttt{T}_{\texttt{Max}}$), occasionally accepting worse solutions for exploration, preventing the algorithm from being trapped in a local optimum. However, as the `T` approaches $\texttt{T}_{\texttt{Min}}$, the algorithm puts more weight on the exploitation. The overall SA search is summarized in the ANNEALER function.

The number of `Steps` highly impacts the results of the search. If there are not enough `Steps` to adequately explore the search space, it might get trapped in a local optimum. We tuned the `Steps` as provided in the Section V-C to prevent such issue. Likewise, an important consideration while determining the initial temperature $\texttt{T}_{\texttt{Max}}$ was to make sure the SA accepts approximately 98% of the new solutions (explore the design space in the early iterations). On the other hand, the final temperature $\texttt{T}_{\texttt{Min}}$ should be low enough such that the new solution does not have too much improvement (SA exploits the good solution founded so far in the later iterations). Since the $\texttt{T}_{\texttt{Max}}$ and the $\texttt{T}_{\texttt{Min}}$ values significantly influence on the result, SCHEDULER function automatically determines the starting values of $\texttt{T}_{\texttt{Max}}$ and $\texttt{T}_{\texttt{Min}}$ that would ensure convergence to a low energy solution. SCHEDULER contains three loops where the two first loops are responsible for estimating $\texttt{T}_{\texttt{Max}}$ and the third loop estimates the $\texttt{T}_{\texttt{Min}}$ value. The RUN function iterates `Steps` times and calculates the number of solutions with fewer ENERGY function (`ImprovementRate`) as well as the number of solutions with no improvement but accepted by the probability of EXP(-Δ / `T` (`AcceptanceRate`)). The goal of the SCHEDULER is to refine the value of $\texttt{T}_{\texttt{(Max, Min)}}$ in order to accept 98% of moves at the beginning and reduce the probability of accepting a new solution at the end of search.

Equation 4 and Equation 5 present the objective functions considered in both search stages. The number of network floating-point operations (*FLOPs*) is considered to indirectly represent the network inference time (latency) in Equation 4. On the other hand, Equation 5 represents the second objective function based on latency prediction model. We devise three different latency predictors for FPGA, GPU and Intel® NCS2 to design customized a CNN architecture for each target device. These latency predictors provide accurate estimations of the inference latency as shown in Appendix G, leading to higher reproducibility of our algorithm as shown in Section VI-F.

$$Energy = \frac{FLOPs}{Accuracy} \tag{4}$$

$$Energy = \frac{Latency\ Prediction}{Accuracy} \tag{5}$$

B. FastStereoNet Practical Proof of Convergence

Generally, SA algorithm comes with a stochastic guarantees to reach a global optimum [9]. We present an empirical evaluation of SA compared to different search algorithms such as random search and LAHC to demonstrate the superior optimization performance of SA. According to [10], [11], [12], the random search can find the best architecture in many applications. However, as shown in the Fig. 4, SA reached the lowest global ENERGY. Overall, experiments show it succeeds to find a feasible solution in a reasonable time.

C. FastStereoNet Complexity Analysis

We analyze the computational complexity of the FastStereoNet based on the *"big-O"* analysis (Appendix C) as well as

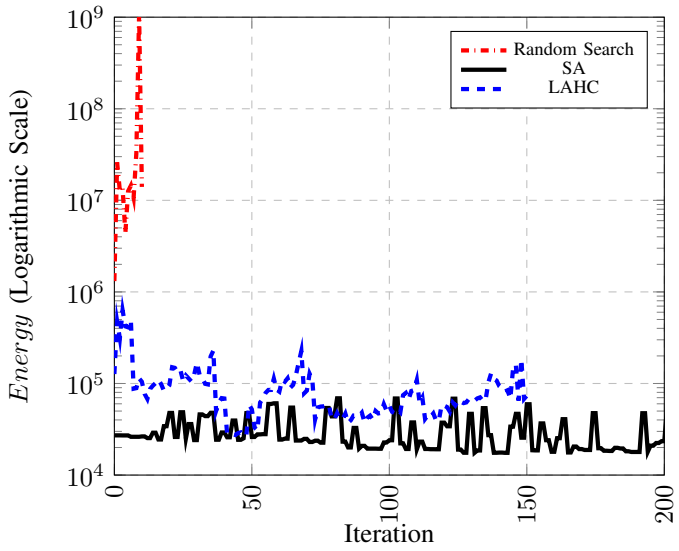


Fig. 4: Comparing the search capability of SA, LAHC, and random search (baseline) with the identical search objective.

the empirical evaluations of the total number of used images during the search (Appendix D). The overall FastStereoNet computational complexity is $O(\texttt{Steps}^2)$, and FastStereoNet uses 120M images in total for designing an efficient architecture that is 9.3× less than 1.15 billion images used by [22].

V. Experimental Setup

In this section, we present the evaluation dataset (Section V-A), hardware specifications (Section V-B), and experimental configuration setup (Section V-C) that are used to test the FastStereoNet framework. The detailed descriptions of the smoothing filters and hardware implementation methods are presented in Appendix E and Appendix F, respectively.

A. Evaluation Dataset

KITTI 2015 is an extensive visual stereo dataset which plays a crucial role in developing several algorithms supporting autonomous systems [54], [25]. The KITTI 2015 dataset is recorded in a city (on highways and rural areas) with a car equipped with a Velodyne HDL-64E LiDAR, GPS, two stereo cameras (color and gray-scale), and inertial sensors [55]. KITTI 2015 includes 200 training and 200 test RGB stereo-pairs with the dimension of 1242×375 pixels. D1-all is an estimation metric in which a pixel is correctly estimated if the difference between the value of a ground-truth pixel and corresponding estimation value is less than 3% (or <5%). KITTI 2015 reports both non-occluded (Noc) pixels and pixels with available ground-truth. The Noc evaluation excludes all pixels falling outside the image surface while pixels occluded by objects within the same image could not be reliably estimated fully automatically because of the laser scanner properties [55]. Therefore, the Noc evaluations have higher accuracy. Some studies use the end-point-error (EPE) as the disparity error metric, however, the KITTI evaluation server[1] only reports the D1-all test set error measurement.

[1]http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

B. Hardware Specifications

Table II presents specifications of utilized hardware devices.

TABLE II: Hardware Specification.

Device	Specification
GPU	NVIDIA® GTX 1080ti (2.5 GHz)
GPU Memory	11 GB
GPU Compiler	cuDNN Version 10.0
Training System Memory	64 GB
CPU	Intel® Mobile Processor (Core i7-7820 HQ)
CPU Compiler	TensorFlow Compiled on Intel®SSE4 Instruction Set Intel® OpenVINO™ Version 2020.1 [56]
FPGA Instance	Amazon EC2 (Zynq UltraScale+)
Intel® NCS2 Accelerator	Movidius™ Myriad™ Vision Processing Unit
Intel® NCS2 Compiler	Intel® OpenVINO™ Version 2020.1

C. Experimental Configuration

Some recent studies use partial training to accelerate the evaluation process [36], [2]. However, in this work, we do not use any search proxies such as accuracy predictors to provide accurate evaluations. The candidates are trained using AdaGrad stochastic gradient descent [15]. AdaGrad adapts the gradient-based on historical information similar to momentum-based gradient descent. Table III summarizes the setup of experiments utilized for the full training and search steps. Finally, we used TensorFlow Version 1.14 to develop the FastStereoNet framework.

TABLE III: The configuration setup of the training and search procedure.

Full-Training Parameters	Value
Activation Function	Relu
# Epochs	1000
Batch Size	128
Disparity Range	128
Optimizer	AdaGrad
Learning Rate (lr)	0.01 and if (epoch⩾2400) then lr = lr/5 after each 800 epochs
Search Parameters	**Value**
Total Steps ($\texttt{Steps}_{\texttt{total}}$)	200
Default Max Temperature ($\texttt{T}_{\texttt{Max}}$)	25000
Default Min Temperature ($\texttt{T}_{\texttt{Min}}$)	2.5
`Steps`	10
`MaxRateAnnealing` in the ANNEALER function	20
`IdleFraction` in the LAHC function	0.02
{`MinRate`, `MaxRate`} in the LAHC function	{10, 30}
{$\texttt{MaxRateT}_{\texttt{Min}}$, $\texttt{MaxRateT}_{\texttt{Max}}$} in the SCHEDULER function	{5,10}

VI. Results

We evaluate the performance of FastStereoNet on various resource-limited devices. Disparity estimation performance, latency, search speed, search convergence, reproducibility, disparity refinement, frequency selection of dominant operations (Appendix I), and hardware implementation results are recorded as the evaluation metrics. We solved the NAS problem with different search scenarios: Late Acceptance Hill Climbing (*LAHC*), Random Search (*RS*), Simulated Annealing without using transfer learning (*SA*), Simulated Annealing with using Transfer Learning (*SA+TL*), Late Acceptance Hill Climbing and Simulated Annealing without using transfer learning (*LAHC+SA*), and Late Acceptance Hill Climbing and Simulated Annealing with using Transfer Learning (*LAHC+SA+TL*).

TABLE IV: Comparing the FastStereoNet results with state-of-the-art methods on KITTI 2015. Unsuccessful implementation are shown in the red cells (mainly due to the limited on-chip memory or OpenVINO™ limited supporting operations).

Architecture	FLOPs ($\times 10^6$)	Search Method	Network Compression Rate($\times$)†	NID ($\times 10^6$)	$\frac{\text{Accuracy}}{\text{FLOPs}}$ ($\times 10^6$)	Search Cost (GPU Days)⋈	Error (%) Without Quantization	Intel® NCS2 Inference Time (Sec.)∗
DenseDisp [2]	1.56	Meta-heuristic	102	89.3	58.98	2	7.99 (D1-all)	0.626 (0.017≠)
AutoDispNet-BOHB-CSS-ft† [1]	160	RL	1	0.88	-	42	OUT_OF_MEMORY‡	
DenseMapNet [57]	-	Hand-Crafted	-	-	-	-	2.52 (EPE)	0.45
Vid2Depth [58]	-	Hand-Crafted	-	-	-	-	0.163 (Abs. Rel.)	0.276
GC-Net [59]	-	Hand-Crafted	-	27.75	-	-	OUT_OF_MEMORY‡	
Content-CNN [24]	2	Hand-Crafted	80	13.6	47.73	-	4.54 (D1-all)	0.7
GA-Net-deep [9]	-	Hand-Crafted	-	-	-	-	OUT_OF_MEMORY‡	
PSM-Net [10]	30	Hand-Crafted	5.3	6.1	-	-	OUT_OF_MEMORY‡	
DispNet-CSS-ft∓ [60]	195	Hand-Crafted	0.8	0.84	-	-	OUT_OF_MEMORY‡	
FastStereoNet (FLOPs)	2.08	Meta-heuristic	76.9	68.24	44.94	8	6.51 (D1-all)	0.64 (0.028≠)
FastStereoNet (NCS2)	3.6	Meta-heuristic	44.4	40.07	26.6	8	4.22 (D1-all)	0.64 (0.028≠)

† The baseline for comparing the compressing rate over the FLOPs metric.
∓ Reported in [1].
∗ The results are compiled with Intel® OpenVINO™.
‡ Undesired state which happens whenever the Intel® NCS2 on-chip memory cannot be allocated due to the huge network size.
⋈ All the methods used 1× NVIDIA® GTX 1080ti for evaluating the candidates.
≠ Average computation time (kernel time) for 10000 times re-running network inference.

A. Disparity Estimation Performance

Table IV compares the FastStereoNet optimized architecture with the other cutting-edge architectures regarding the D1-all accuracy, Intel® NCS2 inference time, FLOPs, and the search cost evaluation metrics. We consider end-to-end latency (data transfer time + computation time) as the evaluation inference metric. Batch size is equal to 1 in all the experiments. We believe taking inference time, represented in second(s), is not reliable as the only metric for comparing the implementation efficiency of two different networks. The reasons come from the fact that 1) the inference time even on the same device depends on various factors, such as the learning API (Torch [61], TensorFlow [44], etc.), compiler settings, and hardware acceleration libraries (NVIDIA® cuDNN, Intel® OpenVINO™, etc.); and 2) the hardware implementation details are not usually reported in many studies. Therefore, we also report *network compression rate*, $\frac{Accuracy}{FLOPs}$, and *NID* (Section II-B2) in Table IV as three hardware-independent alternative metrics. The architecture with the highest latency-error trade-off, optimized by the *LAHC+SA+TL* scenario, is reported as the final optimized FastStereoNet architecture. Fig. J.1 in Appendix J illustrates the FastStereoNet optimized architecture for each device.

FastStereoNet obtains 95.78% accuracy with 640ms total inference time on the Intel® NCS2 accelerator, 2.39M parameters, and 3.6M FLOPs. In comparison with the *AutoDispNet-BOHB-CSS-ft* NAS method, FastStereoNet presents 44.4× more network compression rate while delivering a comparable accuracy (less than 2.05% accuracy loss). It is worth to mentioning that unlike *AutoDispNet-BOHB-CSS-ft*, we do not use any training optimization technique to improve the accuracy of disparity estimation. Also, FastStereoNet can be successfully implemented on the Intel® NCS2 accelerator, while *AutoDispNet-BOHB-CSS-ft* fails to be implemented in the Intel® NCS2 accelerator due to huge memory footprint. In terms of search time, although FastStereoNet directly searches a huge space with the minimum size of 2×24^{10} candidates, FastStereoNet is still 5.25× faster than *AutoDispNet* which is a cell-based search that usually take shorter time to find a solution. Compared to DenseDisp [2], FastStereoNet provides 3.77% higher accuracy as it uses a more complex design space. DenseDisp yields 4× faster search compared to FastStereoNet.

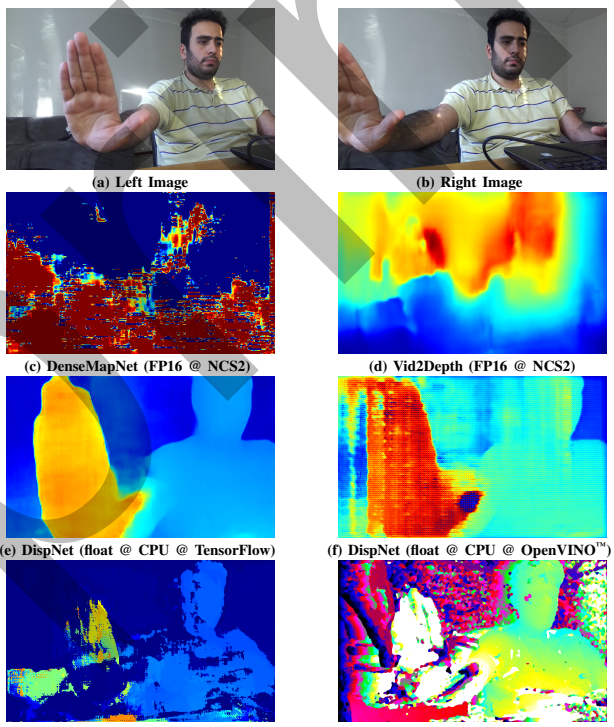


Fig. 5: Illustrating the output of different studied disparity estimators: (a) left image, (b) right image,(c) DenseMapNet (FP16 @ NCS2), (d) Vid2Depth (FP16 @ NCS2), (e) DispNet (float @ CPU @ TensorFlow), (f) DispNet (float @ CPU @ OpenVINO™), (g) GC-Net (float @ CPU), and (h) FastStereoNet (FP16 @ NCS2). FastStereoNet is the only solution that yields clear disparity perception on Intel® NCS2.

However, this comes from the fact that DenseDisp trains the candidate architectures for few epochs to estimate the accuracy while FastStereoNet fully trains the network for all candidates.

Reporting the inference time on the Intel® NCS2 accelerator is motivated by the following three observations: 1) some real-time neural architectures such as DispNet [62] cannot be deployed on some of the resource-limited devices due to their high memory footprint; 2) the cutting-edge learning models use complex operations which are not usually supported by commodity embedded devices, e.g., Intel® NCS2 does not support `GatherNd` operation used by MADNet [63]; and 3)

Most of the resource-limited devices only support quantized operations such as 8-bit floating-point (FP8) or 16-bit floating-point (FP16). However, despite our expectation, the 16-bit quantization decreases the accuracy significantly compared to the full precision implementation (float) as illustrated in Fig. 5.c. As such, the models tailored to high-performance GPUs may not be useful for any resource constrained devices. To the best of our knowledge, FastStereoNet is the only solution that tackles all the challenges mentioned above by providing a clear perception (Fig. 5.h) on the Intel® NCS2 accelerator. FastStereoNet achieves the clear perception with low latency by: 1) considering network inference time as the second search objective yields highly customized architectures for a given target device; 2) using widely supported operations (Table I) makes FastStereoNet favorable for many resource-limited devices; and 3) utilizing an independent disparity refinement which is not quantized during hardware implementation; thus, it refines quantization drawbacks.

B. Estimation Error Across Different Metrics

Table V compares the D1-all disparity accuracy across different error metrics for the model designed for Intel® NCS2. In this experiment, we do not use any disparity refinement techniques. FastStereoNet obtains a 3-pixel stereo error of 7.82%, while Content-CNN [24] achieves 8.97% with 2.9× more parameters. [64] performs slightly better than FastStereoNet on 2- and 3-pixel. However, this is because the main objective of FastStereoNet was to meet the resource constraints while achieving a reasonable performance, whereas [64] solely focuses on improving the accuracy without consideration for the resource constraints.

TABLE V: Comparison of the output of the matching network across different error metrics. The table illustrates D1-all (non-occluded+occluded) error on the KITTI 2015.

Methods without Disparity Refinement	**> 2-pixel**	**> 3-pixel**	**> 4-pixel**	**> 5-pixel**
MC-CNN [65]	16.83	14.12	12.72	11.80
Content-CNN [24]	11.67	8.97	7.62	6.78
Zhou et al. [64]	10.96	7.29	7.28	6.22
FastStereoNet	12.64	7.82	6.94	6.0
Methods with Disparity Refinement	**> 2-pixel**	**> 3-pixel**	**> 4-pixel**	**> 5-pixel**
PSM-Net [24]	3.01	2.32	1.42	1.15
GA-Net-deep [24]	2.79	1.81	1.37	1.1

C. Smoothing Comparison

In this section, we evaluate different disparity refinement filters by employing edge-preserving and smoothing algorithms (Appendix E describes the filters). Table VI shows the 3-pixel error results after applying different filters. The qualitative disparity results of the studied smoothing filters are illustrated in Fig. 6. The yellow circles in Fig. 6 indicate the disparity region with noise and low-confidence matches. Employing smoothing filters shows a notable accuracy improvement (between 0.5% to 3.6%). The cost aggregation encourages local smoothness leading to improve the results slightly. The Median filter eliminates many image noises, but minor isolated noisy areas were remaining (Fig. 6.c). In addition, we observe that MATLAB® smoothing functions followed by Median filter achieves up to 2.6% edge-preserving performance (Fig. 6.c). However, there still exist some low-confidence matches. The Binomial filter is not effective since it works as a linear band-pass filter which only blurs the disparity map. LRCSGM is a robust filter focusing on sub-pixel enhancement. LRCSGM yields the maximum error reduction regarding the raw disparity by 3.6% (as shown in Fig. 6.d).

TABLE VI: Comparison of different smoothing methods applied on the raw output of the FastStereoNet architecture. The table illustrates the 3-pixel error results on the KITTI 2015. The best result is highlighted in green cell.

Smoothing Filter	**DenseDisp [2]**	**FastStereoNet (Ours)**
-	12.4	7.82
CA	11.9	7.27
Median [15×15]	9.9	5.41
LRCSGM	7.9	4.22
MATLAB® `imguidedfilter`	13.0	8.4
MATLAB® `anisodiff2D`	11.2	6.55
MATLAB® `anisodiff2D` + Median [15×15]	9.75	5.24
MATLAB® `sharpen` + Median [15×15]	9.9	5.39
Bilateral + Median [15×15]	10.4	6.01
Binomial [5×5]	13.1	8.7

D. Search Convergence Analysis

Fig. 7 plots the energy function (Equation 4) across search iterations. FastStereoNet finds architectures with a monotonic decrease in the energy, indicating FastStereoNet leads to a higher FLOPs-error trade-off with more search iterations. In addition, results show that *LAHC+SA* and *LAHC+SA+TL*, that begins from better initial conditions from *LAHC*, provided up to 20.1× less energy compared to *SA* and *SA+TL* without *LAHC* initialization.

E. Analyzing Search Methods

Fig. 8.b illustrates the FLOPs-error trade-off for the improved architectures (mutation with improving the objective function) over different search methods. We plot the optimization results with Equation 4 as the baseline objective function. *RS* could not find any improved architectures, meaning that our search space was not the only reason behind the efficiency of FastStereoNet. Although *LAHC* is very fast as it required only 47 iterations to converge (Fig. 8.a), it could not find as efficient neural architecture as it was trapped in local optima. *LAHC+SA+TL* and *SA+TL* provide the best trade-off between FLOPs and error compared to the other methods. However, *LAHC+SA+TL* is 1.8× faster than *SA+TL*, because *LAHC* provided better initialization that led to faster convergence of *SA* (Fig. 8.a). Although *LAHC+SA+TL* has 2.1× slower search convergence compared to *LAHC+SA*, it discovers more improved architectures with a better FLOPs-error trade-off as the transferred weights mechanism led to better solution [43].

F. Reproducibility of the Results

Many works on Neural Architecture Search (NAS) had issues regarding reproducibility due to the innate stochasticity. We demonstrate the reproducibility of FastStereoNet by plotting the improvement with the shades to denote the confidence

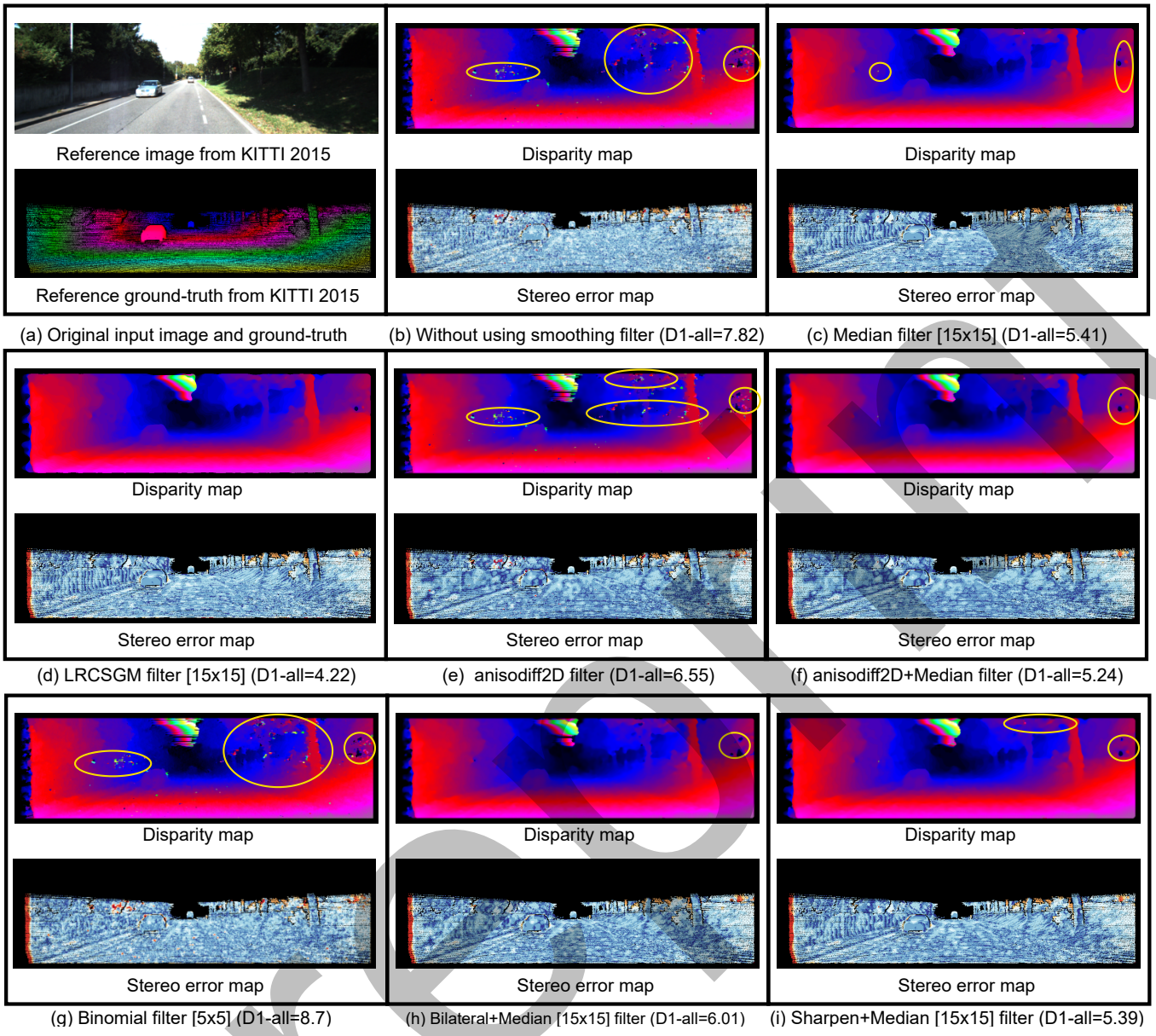


Fig. 6: Visual results of the proposed FastStereoNet framework after applying different disparity refinement filters. The yellow circles indicate the disparity region with noise and low-confidence matches.

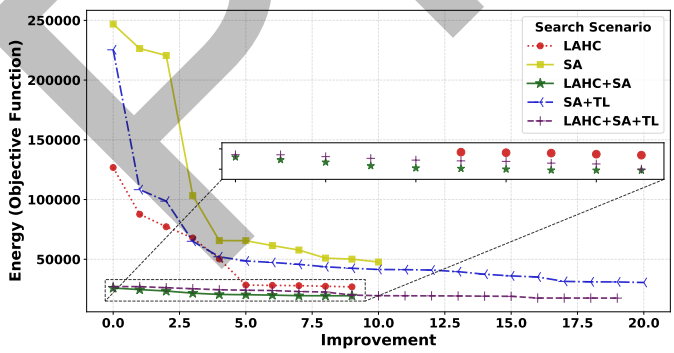
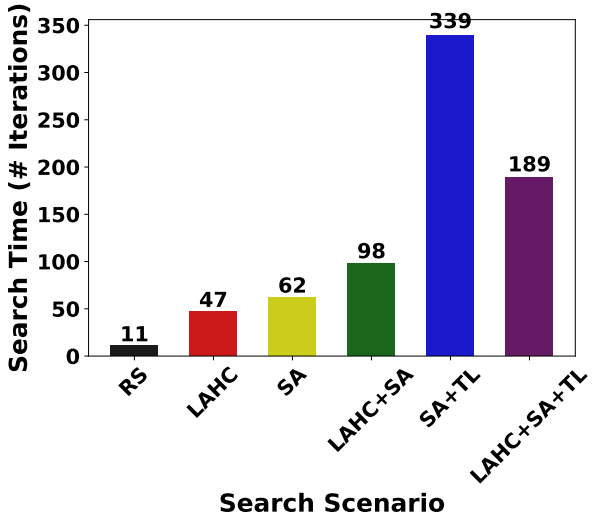


Fig. 7: Convergence of the energy function.

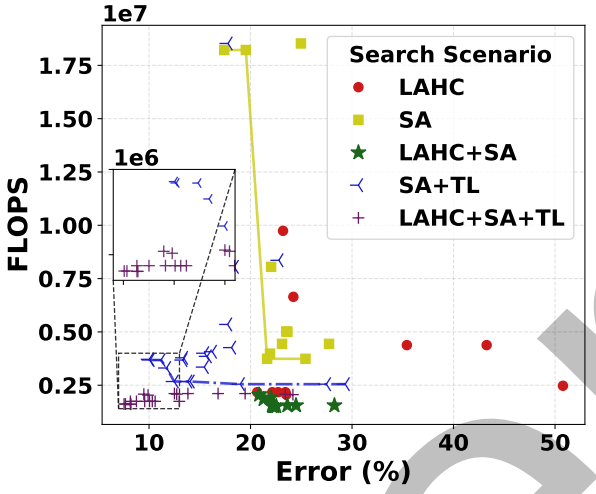
intervals in Fig. 9. Results show that, while the confidence interval is wide in the beginning, all search runs converge to neural architectures with similar energy. Importantly, the search for GPU, FPGA, and Intel® NCS2 were based on latency prediction. Results show that the search based on latency prediction provided higher confidence results with smaller standard deviation during the search and converged significantly faster compared to using FLOPS as the search objective. Appendix H reports the reproducibility of results and correlation coefficients of multiple running FastStereoNet by using the Equation 4 objective function.

G. FPGA Implementation Results

In this section, we discuss the implementation of deploying our optimized network on FPGA. Table VII presents the comparison of the inference time for different disparity estimation



(a)



(b)

Fig. 8: (a) The number of taken iterations to achieve the best candidate over different search scenarios. (b) FLOPs-error trade-off of discovered candidates by different search methods. We also plot the Pareto frontier of each scenario to tell the decision maker the best balanced architecture.

methods implemented on FPGA. We use the number of processed Frame-Per-Second (FPS) as the metric. FastStereoNet provides 2.07× faster inference time compared to Content-CNN as a CNN-based disparity estimation [24]. FastStereoNet provides 11% faster inference time in comparison with the architecture optimized by the FLOPs objective metric indicating that the latency prediction provides better search guidance. In comparison with [14], as an SGM based method, FastStereoNet delivers 3.27× less FPS, but provides 9.68% more accurate estimation. [27] presents cutting-edge results on SGM-based disparity estimation on FPGA by leveraging a heterogeneous device (Xilinx ZCU104 + CPU) leading to achieve considerable performance improvement. Although FastStereoNet fails in comparison with [27], to have a fair comparison, we need to consider that [27] only implements a subset of disparity estimation pipeline on the FPGA (Median filters, SGM, redundancy checks, etc.), while the dynamic tasks with high access to memory are implemented on CPU

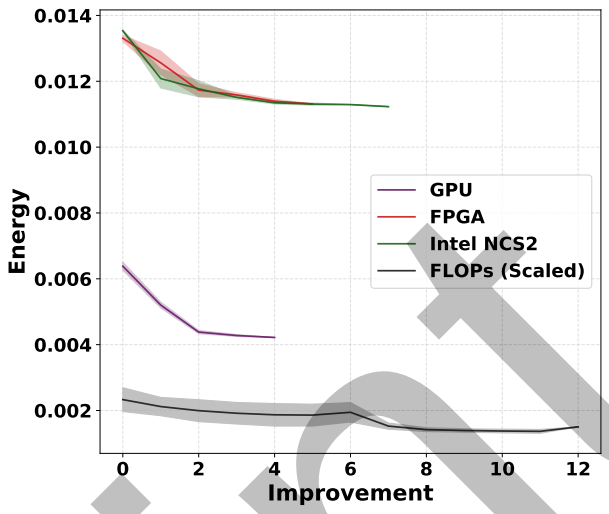


Fig. 9: Illustrating the reproducibility of results. Each line presents the mean value of five times running of FastStereoNet for various devices. Shades represent the confidence interval.

TABLE VII: FPGA implementation results of FastStereoNet compared to state-of-the-art.

Work FPGA Device	Method	Disparity Range	FPS	Accuracy (%) (D1-all)
[14] Zynq ZC706 (CPU+FPGA)	ELAS†[66]	-	9.5	86.1
[27] Zynq ZCU104 (CPU+FPGA)	SGM+ELAS	-	52	91.3‡
Content-CNN [24] Zynq UltraScale+	CNN	128	1.4	95.46
FaststereoNet (FLOPs) Zynq UltraScale+	CNN	128	2.6	93.49
FaststereoNet (FPGA) Zynq UltraScale+	CNN	128	2.9	95.78

† The Efficient Large-Scale Stereo (ELAS), is the fastest triangulation-based disparity estimation algorithm on CPU.

(image transformations, disparity interpolation, etc.).

VII. Conclusion

Directly solving the NAS problem on large scale tasks such as KITTI 2015 is highly expensive, as each candidate takes days to converge. A main disadvantage of evolutionary and RL based methods is that they require enormous computational resources prohibiting their usage in limited budget circumstances. In this paper, we propose a multi-stage search method, named FastStereoNet, which is significantly more efficient in terms of search cost and quality of the results compared to the state-of-the-art. FastStereoNet considers the estimated network inference time along with accuracy as the search objectives to discover resource-efficient architectures. The experiments showed FastStereoNet provides competitive results compared to other alternative approaches.

Acknowledgment

The authors would like to thank Mr. Mohammad Riazati for helping us to get FPGA results. This work was supported by the Swedish Knowledge Foundation (KKS) through DeepMaker and DPAC projects, and VINNOVA through AutoDeep project.

REFERENCES

[1] T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox, "Autodispnet: Improving disparity estimation with automl," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1812–1823.

[2] M. Loni, A. Zoljodi, D. Maier, A. Majd, M. Daneshtalab, M. Sjödin, B. Juurlink, and R. Akbari, "Densedisp: Resource-aware disparity map estimation by compressing siamese neural architecture," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[3] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[4] L. Niu, O. Suominen, M. M. Aref, J. Mattila, E. Ruiz, and S. Esque, "Eye-in-hand manipulation for remote handling: Experimental setup," in *IOP Conference Series: Materials Science and Engineering*, vol. 320, no. 1. IOP Publishing, 2018, p. 012007.

[5] C. Liu, W. Wang, J. Shen, and L. Shao, "Stereo video object segmentation using stereoscopic foreground trajectories," *IEEE transactions on cybernetics*, 2018.

[6] F. Logothetis, R. Mecca, and R. Cipolla, "A differential volumetric approach to multi-view photometric stereo," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1052–1061.

[7] S. Daniel, S. Richard, and H. Heiko, "Middlebury stereo evaluation-version 3."

[8] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan, "Cascade residual learning: A two-stage convolutional neural network for stereo matching," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 887–895.

[9] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, "Ga-net: Guided aggregation net for end-to-end stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 185–194.

[10] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5410–5418.

[11] C. Ahlberg, M. L. Ortiz, F. Ekstrand, and M. Ekstrom, "Unbounded sparse census transform using genetic algorithm," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2019, pp. 1616–1625.

[12] B. H. Ahn, J. Lee, J. M. Lin, H.-P. Cheng, J. Hou, and H. Esmaeilzadeh, "Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices," *arXiv preprint arXiv:2003.02369*, 2020.

[13] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and 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.

[14] O. Rahnama, D. Frost, O. Miksik, and P. H. Torr, "Real-time dense stereo matching with elas on fpga-accelerated embedded devices," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2008–2015, 2018.

[15] D. Zha, X. Jin, and T. Xiang, "An improved global stereo-matching on fpga for real-time applications," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 274–274.

[16] S. Jin, J. Cho, X. Dai Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE transactions on circuits and systems for video technology*, vol. 20, no. 1, pp. 15–26, 2009.

[17] M. Loni, A. Zoljodi, S. Sinaei, M. Daneshtalab, and M. Sjödin, "Neuropower: Designing energy efficient convolutional neural network architecture for embedded systems," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 208–222.

[18] M. Loni, A. Majd, A. Loni, M. Daneshtalab, M. Sjödin, and E. Troubitsyna, "Designing compact convolutional neural network for embedded stereo vision systems," in *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2018, pp. 244–251.

[19] T. Elsken, F. Hutter, and J. H. Metzen, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[20] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[21] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[22] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[23] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[24] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5695–5703.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[26] L. A. Libutti, F. D. Igual, L. Pinuel, L. De Giusti, and M. Naiouf, "Benchmarking performance and power of usb accelerators for inference with mlper*."

[27] O. Rahnama, T. Cavallari, S. Golodetz, A. Tonioni, T. Joy, L. Di Stefano, S. Walker, and P. H. Torr, "Real-time highly accurate dense depth on a power budget using an fpga-cpu hybrid soc," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 5, pp. 773–777, 2019.

[28] X. Dong and 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.

[29] B. Zoph, V. Vasudevan, J. Shlens, and 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.

[30] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SmaSH: One-shot model architecture search through hypernetworks," in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[32] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[33] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: for best speed/accuracy trade-off in neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9145–9153.

[34] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[35] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[36] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocessors and Microsystems*, vol. 73, p. 102989, 2020.

[37] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, 2020.

[38] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evolutionary Computation*, vol. 28, no. 1, pp. 141–163, 2020.

[39] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[40] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, 2019.

[41] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[42] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," in *International Conference on Machine Learning*. PMLR, 2018, pp. 678–687.

[43] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-

scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.
[45] Y. Liu, J.-Q. Ma, T. He, B.-Q. Li, and J.-R. Chu, "Hybrid simulated annealing-hill climbing algorithm for fast aberration correction without wavefront sensor," *Guangxue Jingmi Gongcheng(Optics and Precision Engineering)*, vol. 20, no. 2, pp. 213–219, 2012.
[46] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2390–2401, 2014.
[47] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *European Journal of Operational Research*, vol. 258, no. 1, pp. 70–78, 2017.
[48] H. Szu and R. Hartley, "Fast simulated annealing," *Physics letters A*, vol. 122, no. 3-4, pp. 157–162, 1987.
[49] V. Granville, M. Krivánek, and J.-P. Rasson, "Simulated annealing: A proof of convergence," *IEEE transactions on pattern analysis and machine intelligence*, vol. 16, no. 6, pp. 652–656, 1994.
[50] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 371–385, 2020.
[51] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.
[52] X. Dong, L. Liu, K. Musial, and B. Gabrys, "Nats-bench: Benchmarking nas algorithms for architecture topology and size," *IEEE transactions on pattern analysis and machine intelligence*, 2021.
[53] R. Stuart, N. Peter *et al.*, "Artificial intelligence: a modern approach," 2003.
[54] M. Poggi, F. Tosi, K. Batsos, P. Mordohai, and S. Mattoccia, "On the synergies between machine learning and stereo: a survey," *arXiv preprint arXiv:2004.08566*, 2020.
[55] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070.
[56] N. Smith, "Transitioning from intel® movidius™ neural compute sdk to intel® distribution of openvino™ toolkit," 2019.
[57] R. Atienza, "Fast disparity estimation using dense networks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3207–3212.
[58] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5667–5675.
[59] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 66–75.
[60] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4040–4048.
[61] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.
[62] AlessioTonioni, "Code for "real-time self-adaptive deep stereo" - cvpr 2019 (oral)," 2019. [Online]. Available: https://github.com/CVLAB-Unibo/Real-time-self-adaptive-deep-stereo
[63] A. Tonioni, F. Tosi, M. Poggi, S. Mattoccia, and L. D. Stefano, "Real-time self-adaptive deep stereo," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 195–204.
[64] C. Zhou, H. Zhang, X. Shen, and J. Jia, "Unsupervised learning of stereo matching," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1567–1575.
[65] J. Zbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1592–1599.
[66] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Asian conference on computer vision*. Springer, 2010, pp. 25–38.

Mohammad Loni is a Ph.D. candidate at Mälardalen University (MDU). Currently, he is a visiting researcher at AutoML research group, Leibniz University Hannover, Germany. He is a member of DPAC, AutoDeep and HERO projects. He works on efficient implementation of deep neural networks on embedded platforms using neural architecture search techniques.

Ali Zoljodi is a Ph.D. student at the School of Innovation, Design, and Engineering at Mälardalen University (MDU). He received his M.Sc. degree in software engineering from Shiraz technical University. He works on robust development of computer vision algorithms for autonomous driving applications.

Amin Majd is currently a senior researcher and lecturer at Arcada University of Applied Sciences on smart swarms of autonomous vehicles with a broad background in computer science. He got his first PhD from the University of Turku in 2019 by developing a hybrid meta-heuristic optimization method. The second PhD has obtained in 2021 from Åbo Akademi University by developing the DIANA as safe and efficient navigation of autonomous vehicles.

Byung Hoon Ahn is a Ph.D. candidate at University of California San Diego working under the supervision of Prof. Hadi Esmaeilzadeh. Prior to Ph.D. studies, he worked at Samsung Research as a Software Engineer. His research interests include Computer Architecture, Compilers, and Machine Learning.

Masoud Daneshtalab is currently a Prof. at Mälardalen University (MDU), Adj. Prof. at TalTech, and co-leading the Heterogeneous System research group. He is on the Euromicro board of directors, an editor of the MICPRO journal, and has published over 200 refereed papers. His research interests include HW/SW co-design, reliability, and deep learning acceleration.

Mikael Sjödin received his Ph.D. in computer systems in 2000 from Uppsala University. Since then, he has been working in both academia and in the industry with embedded systems, real-time systems, and embedded communications. In 2006 he joined the Mälardalen University (MDU) faculty as a full professor with a specialty in real-time systems and cyber-physical systems.

Hadi Esmaeilzadeh received his Ph.D. from the Department of Computer Science and Engineering at the University of Washington in 2013. Currently, he is the Halicioğlu Chair in Computer Architecture at University of California San Diego with the rank of Associate Professor. He received the IEEE Young Computer Architect Award in 2018. His research interests include Computer Architecture, Machine Learning, and Data Security.

Supplementary Material

Appendix A
Disparity Estimation Pipeline

In this section, we first present the pipeline of disparity estimation. Then, we review the methods that use triangulation for efficient disparity estimation on resource-limited devices. The stereo vision triangulation is the conventional method for extracting depth information in multiple simultaneous 2D images. The conventional stereo vision algorithm consists of the following four steps [1]: 1) cost initialization, calculating the matching costs for assigning different disparities to different pixels; 2) cost aggregation, spatially aggregating the initial matching costs over a supporting window; 3) disparity optimization, selecting the best disparity for each pixel by minimizing the matching cost; and 4) disparity refinement, post-processing the generated disparity map to preserve the variation of the discontinuous disparity values and removing the mismatches. Fig. A.1 pictures the processing steps of the triangulation-based stereo matching algorithm.

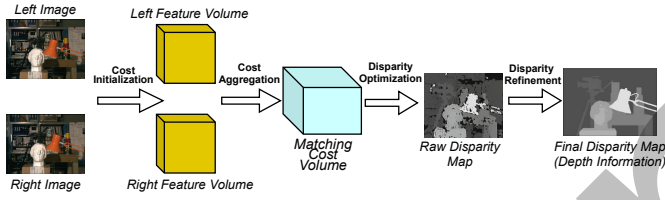


Fig. A.1: The overview of the processing steps of the triangulation-based stereo matching algorithm.

The taken steps in this pipeline depends on the specific methods categorized in three modalities, comprising: 1) local [2], [3]; 2) global [4]; and 3) Semi-Global Matching (SGM) [5]. The global methods try to optimize problem over the entire image, while local methods are restricted to the intensity of surrounding neighborhood information of the pixel under evaluation. In general, the global methods are more accurate, but the local and semi-global methods are more efficient since we can limit the searching range of disparity estimation [6], [7]. Therefore, semi-global methods are preferable for real-time embedded systems. SGM, which is the most widely used stereo matching algorithm, considers the neighboring pixels to extract smooth surfaces, while the resolution of matching ambiguities is much more reliable than local methods.

Appendix B
Simulated Annealing vs Genetic Algorithm

The design space of the target neural architecture search problem is astronomically large (2×24^{10} in the minimum case). To solve this problem using Genetic Algorithm (GA) we have to either 1) consider large population size, usually 10% of the whole design space, with a small number of search iterations; or 2) consider a small population size with a large number of search iterations. However, both of these options are time-consuming. For instance, [8], a cell-based NAS method that uses GA, requires 27 GPU-days to design an optimized architecture for CIFAR-10 dataset. On the other hand, FastStereoNet is a macro NAS method that uses LAHC+SA to design an optimized architecture for KITTI 2015 dataset in only 8 GPU-days. In addition, FastStereoNet leverages the SCHEDULER function to adaptively select the search hyper-parameters (e.g., $\mathtt{T_{Max}}$ and $\mathtt{T_{Min}}$), while [8] needs expert intuition to select search hyper-parameters for any unseen problem.

Appendix C
Traditional Complexity Analysis of FastStereoNet

The FastStereoNet algorithm consists of two sub-functions, including LAHC and SA. Thus, the computational complexity of the FastStereoNet is the summation of the extended LAHC and the extended SA computational complexities. The computational complexity of the extended LAHC depends on the maximum number of search iteration, which is equal to $\mathtt{Steps}\times\mathtt{MaxRate}$. The computational complexity of the MUTATE function and replacement are considered $O(1)$. We also consider the computational complexity of the ENERGY function (training) equal to $O(1)$ to only compute the order of search complexity. Considering the $\mathtt{MaxRate}$ value is constant, $O(\mathtt{Steps})$ is counted as the computational complexity of the extended LAHC. In other words, the computational complexity of the extended LAHC only depends on the maximum number of produced networks.

The SA algorithm contains two main sub-functions, including SCHEDULER and ANNEALER. The worse-case computational complexity of the ANNEALER depends on the maximum number of search iteration which is equal to $\mathtt{Steps}\times\mathtt{MaxRateAnnealing}$. Considering the $\mathtt{MaxRateAnnealing}$ value is constant, the computational complexity of ANNEALER is $O(\mathtt{Steps})$. SCHEDULER employs three independent loops to estimate the value of the $\mathtt{T_{Max}}$ and $\mathtt{T_{Min}}$. In each iteration of these three loops, we $\mathtt{Steps}$ times produce a new solution fulfilled in the RUN function. Since these loops are bounded to a maximum iteration ($\mathtt{Steps}\times\mathtt{MaxRateT}_{(\mathtt{Max},\ \mathtt{Min})}$), the computational complexity of each loop is $O(\mathtt{Steps}^2)$. In the same way, the computational complexity of SCHEDULER is computed as $O(3 \times \mathtt{Steps}^2) = O(\mathtt{Steps}^2)$. Therefore, the computational complexity of SA is $O(\mathtt{Steps})+O(\mathtt{Steps}^2) = O(\mathtt{Steps}^2)$. Similarly, the overall FastStereoNet computational complexity is $O(\mathtt{Steps}) + O(\mathtt{Steps}^2) = O(\mathtt{Steps}^2)$.

Appendix D
Empirical Complexity Analysis of FastStereoNet

Inspired by [13], we evaluate the total number of images used by the search process as a reasonable metric for analyzing the complexity of the proposed search method. We do not consider the transferred weights mechanism in worst-case complexity analysis since it improves search results. Suppose that we train each solution partially with $P = 1000$ epochs to select the best architecture. According to the search configuration, the search process needs $I = 600$ iterations in total. We need $I \times P$ epochs for the end-to-end optimization process. Therefore, we use 120M images (in each P epochs

we evaluate 200 images) in total, that is 9.3× less than 1.15 billion images used by [13].

APPENDIX E
SMOOTHING FASTSTEREONET OUTPUTS

The output of FastStereoNet is a raw predicted disparity of each image location, without applying any disparity refinement. In this paper, we investigate different disparity refinement methods to improve the robustness of the output results. In particular, we use cost aggregation, Median filtering, MATLAB® edge-preserving and smoothing toolbox, as well as Bilateral filtering as a means of noise removal and smoothing. We briefly present these methods in the following.

Cost Aggregation (CA). We leveraged a simple cost aggregation method by performing average pooling over a 5×5 window size.

Median Filtering. Is a popular nonlinear method that preserves sharp edges while tries to effectively suppress impulsive noises by replacing each pixel with the median of neighboring pixels. In this paper, we perform the Median filter over a window size of 15×15 in all the experiments.

MATLAB® Edge-Preserving and Smoothing Filters. MATLAB® provides a set of image enhancement functions that we applied to smooth, sharp, and preserve edges with minimal effort. We exploit the following three filtering functions supported in MATLAB® 2017b.

1) `imguidedfilter`. The guided image filtering is a function that considers the statistics of a region in the corresponding spatial neighborhood of a second image, so-called guidance image, to improve edge-preserving smoothing influence. We used the ground-truth as the guiding image to improve the filtering quality.
2) `anisodiff2D`. The anisotropic diffusion filtering is one of the most popular methods in 3-D reconstruction, image denoising, and stereo matching, which works based on the heat transfer partial differential equation [14].
3) `sharpen`. In general, the sharpening process is a high pass filter that increases the contrast between bright and dark pixels. In this paper, we applied the Median filter (15×15) on the output of a `sharpen` function to smooth the disparity values between adjacent depth-continuous pixels.

Bilateral + Median Filtering. The bilateral filter is a nonlinear image smoothing noise-reducing and edge-preserving filter. The bilateral filter swaps the intensity value of each pixel with the weighted intensity average of its neighbourhood pixels. In this paper, we applied the Median filter (15×15) on the output of the bilateral filter to achieve a higher level of smoothness.

Binomial Filtering. The binomial filter is a statistical smoothing filter based on a kernel filed with binomial weights. The binomial filter does not require multipliers and thus can be deployed efficiently in resource-limited hardware. However, the binomial filter could not enhance disparity refinement performance since it is a band-pass filter which blurs the edges.

Left-Right Consistency + Speckle + Gap Interpolation + Median (LRCSGM) [2]. The algorithm performs the following four steps: 1) Left-right consistency: that requires switching target and reference images in the matching and remove inconsistent matches (the most challenging step); 2) Speckle filter: check whether or not there is a sufficient number of pixels within a window of similar disparity; 3) Gap interpolation: fill non-confident values with the disparity of closes valid disparity; 4) Median filter: the final noise removal function. According to our experimental results, the LRCSGM filter provides the maximum smoothing accuracy.

APPENDIX F
HARDWARE IMPLEMENTATION DETAILS

This section presents steps to deploy a CNN architecture on the target hardware devices.

FPGA. We utilize the Xilinx High-Level-Synthesis (HLS) tool to automatically deploy a CNN architecture represented in Python TensorFlow to FPGA. The main reason for employing HLS for developing the toolchain is that HLS allows neural network designers who actually have little knowledge about the underlying hardware and digital design. However, HLS tools usually only support OpenCL, C or C++, while neural network designers typically use high-level APIs to describe their networks such as TensorFlow, or PyTorch. To solve this challenge, we leveraged DeepHLS [16] conversion tool to convert TensorFlow to ANSI C. In addition to the conversion, DeepHLS supports conversion result verification. Fig. F.1 pictures the overall flow of the toolchain. This process's input is a trained CNN in TensorFlow plus the test data used in the CNN design phase. The main steps of Fig. F.1 are:

1) **Preprocessing.** Generate the memory dump of the test data and network weights and biases.
2) **Synthesizable C code generation.** In the first step, we process the TensorFlow to extract the specifications of each layer. Next, the output code generation begins. In fact, we use a library of C templates for generating the output code.
3) **Testbench.** Test and simulate the generated C code from the earlier step to check the correctness of code. If the data type is exactly the same as what was used in the TensorFlow design phase, the accuracy must be the same. In this paper, we use the 16-bit fixed-point data type for the FPGA implementation.

GPU. TensorFlow is able to automatically deploy a CNN on GPU by utilizing NVIDIA® CUDA® Deep Neural Network library™ (cuDNN). cuDNN is a GPU-accelerated library of primitives for DNNs. cuDNN provides highly optimized implementations of routines that frequently occur in DNN applications. We used cuDNN version 10.0 to perform experiments.

Intel® NCS2. The Myriad 2 Vector Processing Unit (VPU) is designed as a 28-nm co-processor that provides high-performance tensor acceleration with less than 1W chip dissipates [17] and 600MHz nominal frequency. The Intel® Neural Compute Stick 2 (NCS2) is an implementation of the Myriad 2 VPU. Intel® NCS2 uses the OpenVINO™ toolkit to facilitate faster inference. Intel® NCS2. The programming interface supports Python and C/C++. To perform inference on the device, the programming API follows a set of operations that

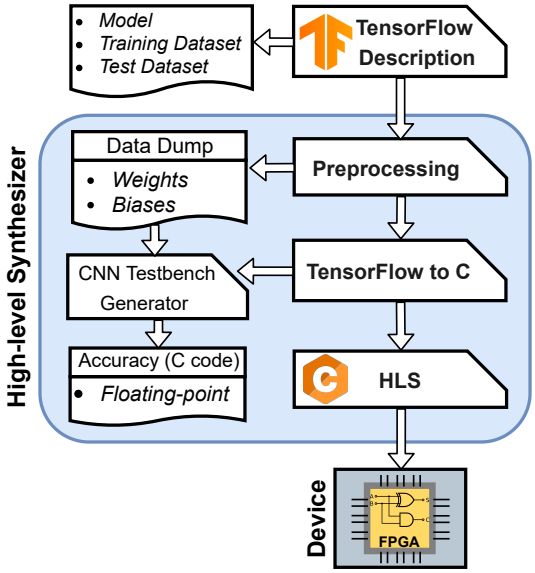


Fig. F.1: The bird's eye view of the DeepHLS toolchain.

is a non-blocking MPI. Thus, a load operation transfers the input and prepares the NCS2 for execution in the first step. Then, a wait operation blocks the process on the host until finishing the execution on the NCS2.

Appendix G
Latency Prediction

Inference time (latency) is the second objective for designing resource-efficient CNN architectures. Optimizing CNN based on direct latency measurement instead of FLOPs can better represent hardware traits [21]. However, direct latency measurement prolongs the design time (up to 20 minutes for measuring the FPGA latency of each candidate [18]). Therefore, direct latency measurement is not ideal for scalable NAS. On the other hand, several studies optimize CNN architecture based on FLOPs [36], [2] or direct performance measurement [17]. Nevertheless, FLOPs cannot capture the desired hardware characteristics [20]. To demonstrate this claim, we compared the latency of a network on two different hardware devices. Fig. G.1 illustrates the measured latency surface of a 3×3 convolution operator with different input and output channel sizes on an Intel® core i5-3210m 2.5 GHz and a Xilinx ZCU104 FPGA. Intel® processor and Xilinx ZCU104 have different latency surface area over network FLOPs.

Instead of direct measurement, we build a regression model to estimate network inference time. For a network architecture with a sequence of operations, the expected inference time of the network can be expressed with the summation of the network operations [19]:

$$EL[latency_{network}] = \Sigma_i EL[latency_i] = \Sigma_i^{LP(o_i)} \quad (6)$$

where $EL[latency_i]$ is the expected latency of the i^{th} operation, LP(o_i) denotes the predicted latency of the operation i^{th}. Accordingly, we need to predict the latency of each network operation. Therefore, the inputs of the latency prediction model include: (a) input and output feature map

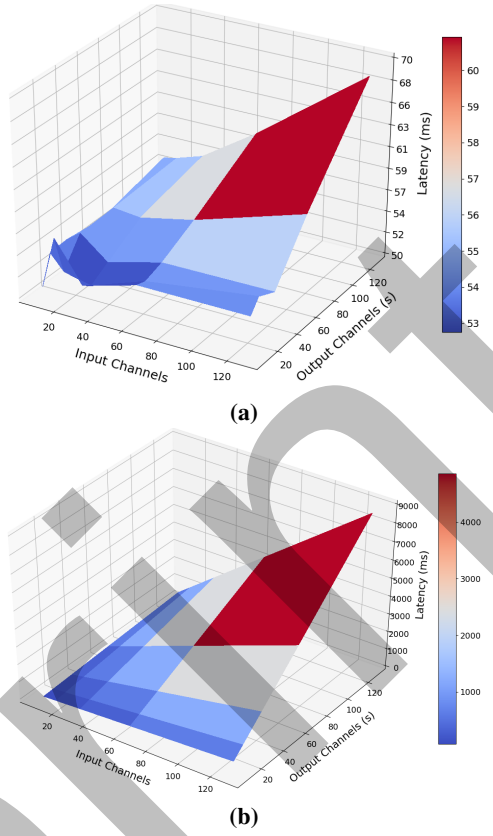


Fig. G.1: Latency vs. #Channels for a 3×3 convolution on an input image size of 56×56 and stride 1 on (a) Intel® core i5-3210m 2.5 GHz and (b) Xilinx ZCU104 FPGA. Blue (red) color indicates low (high) latency.

size, (b) type of the operator, (c) kernel size, and (d) stride size. We randomly sampled 105 architectures from the search space, where 80% of the samples are used to build the latency model, and the rest for testing the model. Fig. G.2 to Fig. G.4 compare several regression models for predicting the latency of GPU, FPGA, and Intel® NCS2, respectively. The regression model with the minimum mean-squared-error (MSE) has been selected as the best prediction model. The results show that there is a strong correlation between directly measured latency and the predicted latency on the test set, suggesting that the proposed latency prediction model can be used to replace the expensive real hardware latency measurement.

Appendix H
Reproducibility of Results

Fig. H.1.a illustrates the reproducibility of results for the baseline scenario using FLOPs-based search method with the same solution as the initial point. The average standard deviation (STDEV) of reproducing the results is 14.2%. The variation of STDEV in the first iterations is up to 36.0% indicating that we explore the design space in early search steps, while we exploit the space in the later search steps with ≃10% STDEV. Fig. H.1.b presents the Pearson correlation coefficients of multiple FastStereoNet runs. Results show that there is a strong relationship between the several times running FastStereoNet with $Pearson - r$ =0.94 on average.

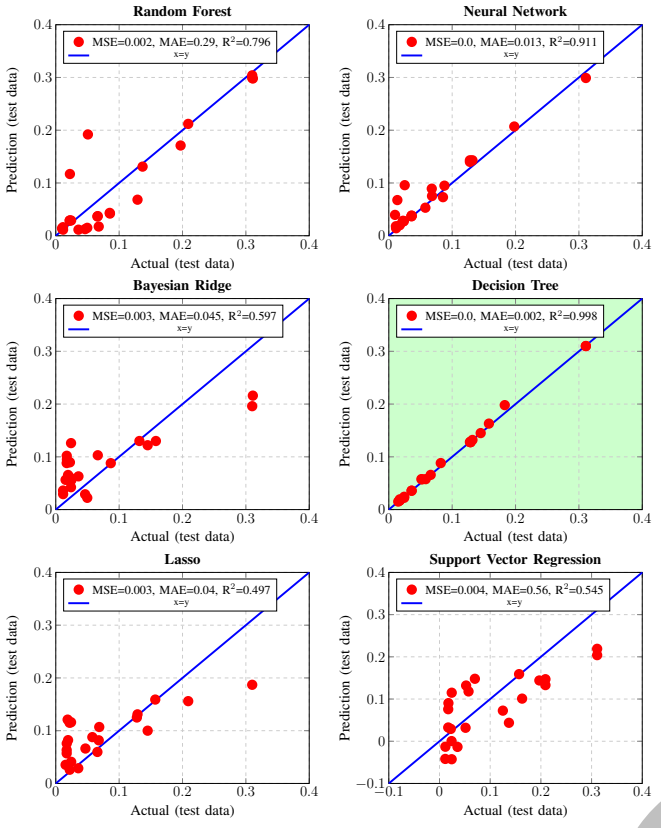


Fig. G.2: Comparison of different GPU latency prediction models. All values are in millisecond. The green figure is the best model.

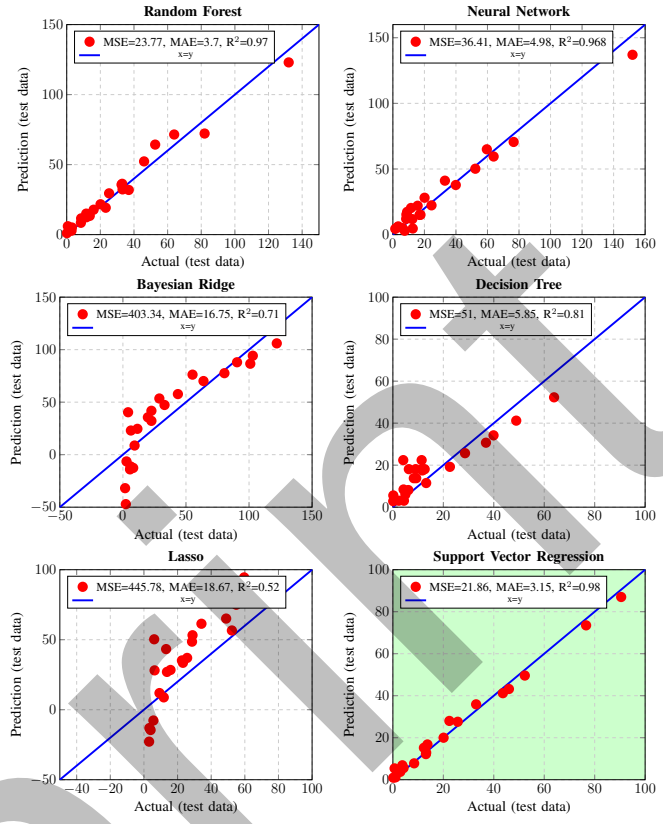


Fig. G.3: Comparison of different FPGA latency prediction models. All values are in millisecond. The green figure is the best model.

Appendix I
Analyzing Frequency Selection of the Dominant Operations

Fig. I.1 presents the dominance pattern of each node operation (method *LAHC+SA+TL*) in the mutated architectures over proceeding the search iterations. The kernel with size 5×5 and the *Same* padding (*Same:5×5*) is the dominant operation in the first random initial candidate (iteration = 1). However, by proceeding the search iterations, the kernel with size 3×3 and the *Valid* padding (*Valid:3×3*) shows promising results by occupying around 70% of nodes after 100 iterations. It means that FastStereoNet favors to select *Valid:3×3* as a superior operation since a small kernel size can extract the tiny features of the image which is extremely important for finding corresponding pixels in two stereo images.

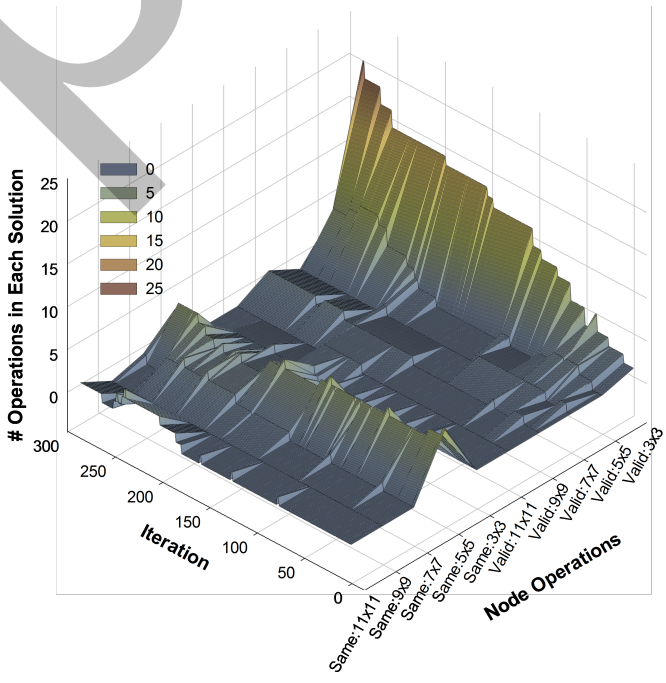


Fig. I.1: Analyzing dominance pattern of each node operation (Table I) in the mutated architectures over search proceeding.

Appendix J
Illustration of the FastStereoNet Optimized Architecture

Fig. J.1 illustrates efficient architectures designed by FastStereoNet for GPU, FPGA, Intel® NCS2 latency predictors, and FLOPs-based objective functions.

References

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," International journal of computer vision, vol. 47, no. 1-3, pp. 7–42, 2002.

[2] C. Ahlberg, M. L. Ortiz, F. Ekstrand, and M. Ekstrom, "Unbounded sparse census transform using genetic algorithm," in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2019,

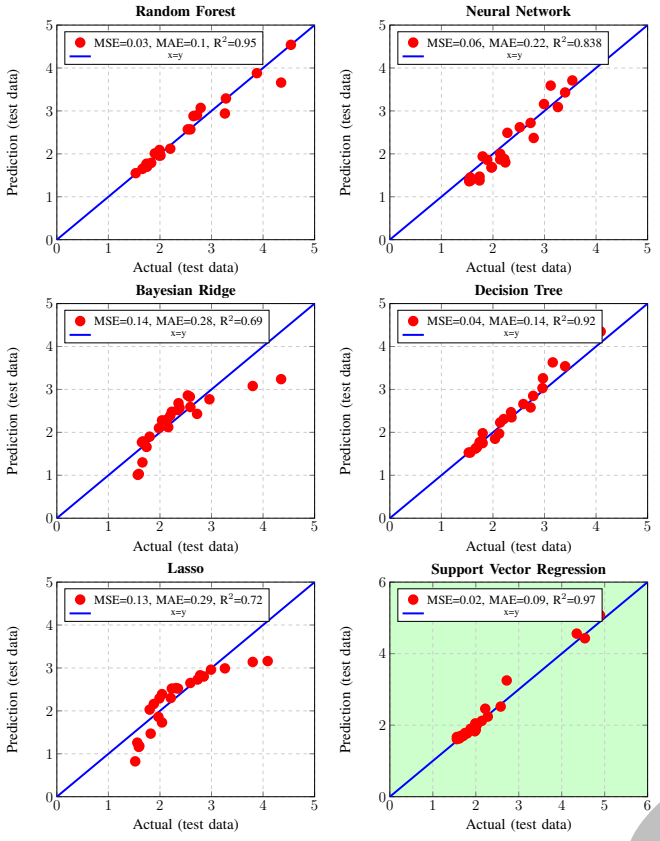


Fig. G.4: Comparison of different Intel® NCS2 latency prediction models. All values are in millisecond. The green figure is the best model.

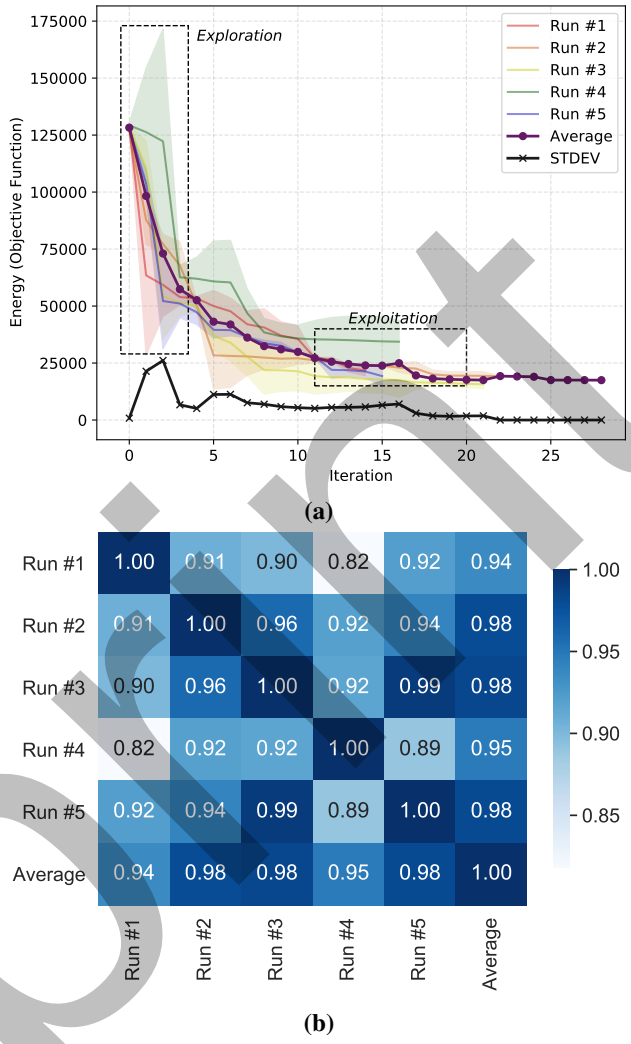


Fig. H.1: (a) Illustrating the reproducibility of the results for the FLOPs-based search. The purple line is the mean value of five times running of FastStereoNet. (b) Correlation coefficients of several times running the FastStereoNet framework.

pp. 1616–1625.

[3] D. Chen, M. Ardabilian, and L. Chen, "A fast trilateral filter-based adaptive support weight method for stereo matching," IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 5, pp. 730–743, 2014.

[4] R. A. Hamzah and H. Ibrahim, "Literature survey on stereo vision disparity map algorithms," Journal of Sensors, vol. 2016, 2016.

[5] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," IEEE Transactions on pattern analysis and machine intelligence, vol. 30, no. 2, pp. 328–341, 2007.

[6] F. Schumacher and T. Greiner, "Matching cost computation algorithm and high speed fpga architecture for high quality real-time semi global matching stereo vision for road scenes," in 17th International IEEE Conference on Intelligent Transportation Systems (ITSC). IEEE, 2014, pp. 3064–3069.

[7] H. Laga, L. V. Jospin, F. Boussaid, and M. Bennamoun, "A survey on deep learning techniques for stereo-based depth estimation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.

[8] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," IEEE Transactions on Evolutionary Computation, 2020.

[9] V. Granville, M. Krivanek, and J.-P. Rasson, "Simulated annealing: A proof of convergence," IEEE transactions on pattern analysis and machine intelligence, vol. 16, no. 6, pp. 652–656, 1994.

[10] A. Yang, P. M. Esperanc¸a, and F. M. Carlucci, "Nas evaluation is frustratingly hard," arXiv preprint arXiv:1912.12522, 2019.

[11] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in Uncertainty in Artificial Intelligence. PMLR, 2020, pp. 367–377.

[12] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," The Journal of Machine Learning Research, vol. 13, no. 1, pp. 281–305, 2012.

[13] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 19–34.

[14] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," IEEE Transactions on pattern analysis and machine intelligence, vol. 12, no. 7, pp. 629–639, 1990.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." Journal of machine learning research, vol. 12, no. 7, 2011.

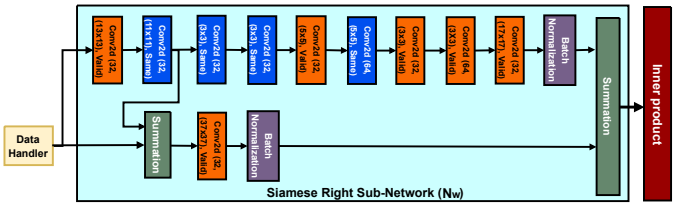
[16] M. Riazati, M. Daneshtalab, M. Sjodin, and B. Lisper, "DeepHLS: A complete toolchain for automatic synthesis of deep neural networks to FPGA," in ICECS 2020 - 27th IEEE International Conference on Electronics, Circuits and Systems, Proceedings, 2020.

[17] D. Moloney, B. Barry, R. Richmond, F. Connor, C. Brick, and D. Donohoe, "Myriad 2: Eye of the computational vision storm," IEEE Hot Chips 26 Symposium (HCS), pp. 1-18, 2014.

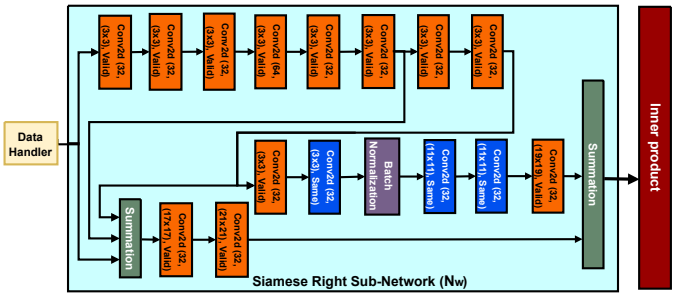
[18] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, and P. Vajda, "Chamnet: Towards efficient network design through platform-aware model adaptation," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11398-11407, 2019.

[19] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," arXiv preprint arXiv:1812.00332, 2018.

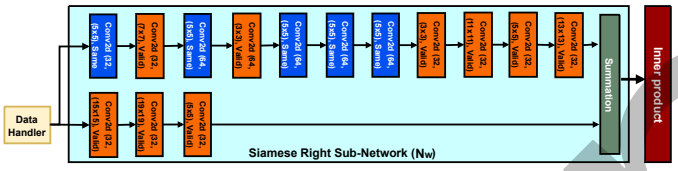
[20] H. Bouzidi, H. Ouarnoughi, S. Niar, and A. A. E. Cadi, "Performance prediction for convolutional neural networks on edge GPUs," in Proceed-



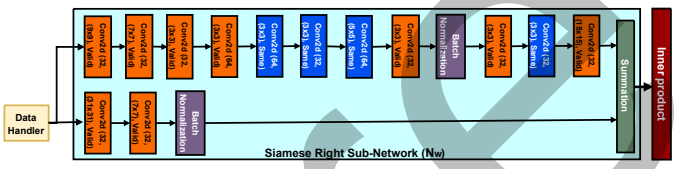
(a) The FastStereoNet best architecture designed for GPU.



(b) The FastStereoNet best architecture designed for FPGA.



(c) The FastStereoNet best architecture designed for Intel® NCS2.



(d) The best architecture with FLOPs-based objective function (Equation 4).

Fig. J.1: The *Conv2d(α, (β, β), Valid/Same)* stands for a convolutional layer with α filters, kernel size of β, and Valid/Same padding.

ings of the 18th ACM International Conference on Computing Frontiers, pp. 54-62, 2021.

[21] 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/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820-2828, 2019.