DenseDisp: Resource-Aware Disparity Map Estimation by Compressing Siamese Neural Architecture

Mohammad Loni*†, Ali Zoljodi‡, Daniel Maier*, Amin Majd§, Masoud Daneshtalab†,
Mikael Sjödin†, Ben Juurlink*, Reza Akbari‡
* Technische Universität Berlin, Germany {daniel.maier, b.juurlink@tu-berlin.de}
‡ Shiraz University of Technology, Iran, {a.zoljodi, akbari}@sutech.ac.ir
† School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
{mohammad.loni, masoud.daneshtalab, mikael.sjodin}@mdh.se
§ Department of Information Technology, Åbo Akademi University, Turku, Finland, {amajd@abo.fi}

***Abstract*—Stereo vision cameras are flexible sensors due to providing heterogeneous information such as color, luminance, disparity map (depth), and shape of the objects. Today, Convolutional Neural Networks (CNNs) present the highest accuracy for the disparity map estimation [1]. However, CNNs require considerable computing capacity to process billions of floating-point operations in a real-time fashion. Besides, commercial stereo cameras produce huge size images (e.g., 10 Megapixels [2]), which impose a new computational cost to the system. The problem will be pronounced if we target resource-limited hardware for the implementation. In this paper, we propose *DenseDisp*, an automatic framework that designs a Siamese neural architecture for disparity map estimation in a reasonable time. DenseDisp leverages a meta-heuristic multi-objective exploration to discover hardware-friendly architectures by considering accuracy and network FLOPS as the optimization objectives. We explore the design space with four different fitness functions to improve the accuracy-FLOPS trade-off and convergency time of the *DenseDisp*. According to the experimental results, DenseDisp provides up to 39.1x compression rate while losing around 5% accuracy compared to the state-of-the-art results.**

***Index Terms*—Stereo Vision, Deep Learning, Multi-Objective Optimization, Neural Architecture Search**

I. Introduction

Stereo cameras are multi-modal vision sensors that are capable to simultaneously extract relative position of 3D objects, color, and luminance. Therefore, stereo cameras are attractive for many applications such as dynamic perception of surrounding environment in autonomous systems. CNNs form the skeleton of visual recognition, image segmentation, and decision making due to providing high accuracy, and ease of usability. CNNs are favorable for stereo vision tasks when the classification accuracy is considered. However, processing CNN-based stereo vision needs huge computing capacity which is extremely challenging for real-time constraints and resource-limited hardware. Reasons are ever-evolving nature of CNNs, containing up to billions of floating-point operations, and producing high-resolution images by modern stereo cameras.

Many efforts have been devoted to improve the implementation efficiency of CNNs [2], [3] by tweaking hyper-parameters of a CNN architecture because optimizing the CNN architecture strongly effects accuracy, inference time, generalization proficiency and memory footprint [3]. Thus, several architecture and training hyper-parameters, such as network activation functions and learning rate, should be tuned in advanced. However, manual optimization is inefficient due to requiring expertise and considerable trial-and-error.

Recently, Neural Architecture Search (NAS) [4], [5] shows promising results in relieving us from manual tweaking hyper-parameters. Unfortunately, most of the NAS methods have huge computing demands [6], [2] leading them to only be applicable on small-scale classification tasks with few training hours, e.g., CIFAR [7]. Several works proposed different techniques to boost the efficiency of NAS such as HyperNet [8], [9], weight sharing [10], accuracy prediction [3], and network transformation [11], [12]. Although these techniques have some advantages (see Section II), complex architectures with large-scale datasets, such as Siamese neural network, is still neglected in research studies.

In this paper, we introduce *DenseDisp*, a multi-objective NAS framework that discovers hardware-friendly architectures by considering network accuracy and total number of floating-point operations (FLOPS) in the entire network as the exploration objective. The reason of considering network FLOPS as an implicit representation of hardware resources is the existing strong correlation between network FLOPS and network inference time ($R^2 = 0.8888, pvalue < 0.0015$) and network energy consumption ($R^2 = 0.9641, pvalue < 0.0001$) [13], [3]. DenseDisp *directly* explores the Siamese architectural space (see Section III-A) by leveraging Simulated Annealing (SA) as a fast, but cost-efficient meta-heuristic exploration method. The main reason of DenseDisp's performance is the single solution based nature of SA, while for example the genetic algorithm is relatively slow due to a population based optimization. However, the quality of our explorations is highly dependent on the SA fitness function. To this end, we propose

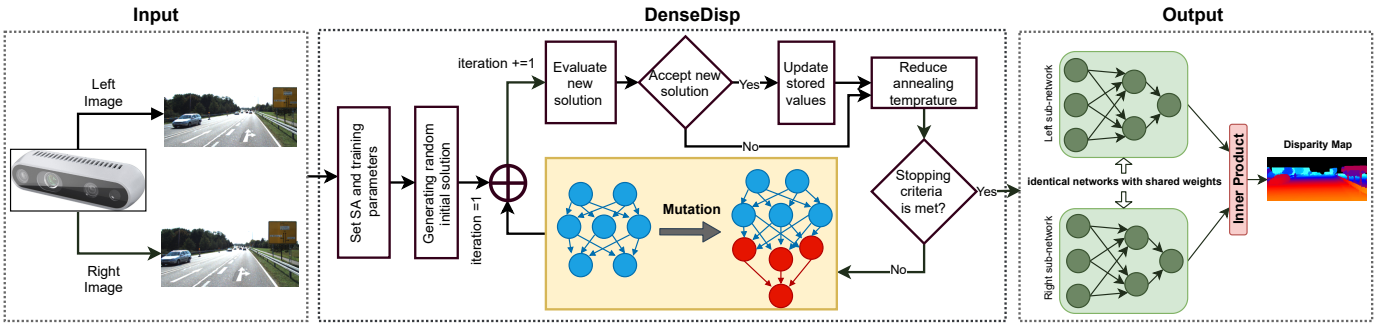

Fig. 1: A bird's-eye view of the DenseDisp framework. The DenseDisp block represents the flow-chart of SA as the exploration algorithm.

four different fitness functions to assess their impact on the exploration speed and discovering efficiency. Fig. 1 depicts an overview of the DenseDisp framework. The input of the DenseDisp consists of two synchronized stream of images received from the stereo camera, while the output is a dense Siamese architecture to be executed on a resource constraint hardware in real-time.

Paper Contributions. The main contributions of this paper is listed below;

- We introduce a fast multi-objective exploration framework, named DenseDisp, that designs Siamese architectures for large-scale tasks in a reasonable time (2 end-to-end GPU-days).
- We propose a dynamic design space by coding the architecture as a list of operational nodes with a variable size (the minimum size of design space is 20^{10}).
- DenseDisp considers the accuracy and the network FLOPS as the exploration objectives. DenseDisp is able to optimize an architecture for a wide range of hardware platforms due to considering a hardware-independent objective (network FLOPS).
- Finally, we compare DenseDisp with state-of-the-art disparity estimation methods to investigate which method provides higher accuracy-FLOPS trade-off with less exploration time. According to the experimental results, DenseDisp has a positive impacts on the landscape of optimizing high accurate disparity estimation for the real-time applications on resource-limited platforms. For example, autonomous robots with limited battery life are among the applications that potentially gain from the DenseDisp framework.

II. Related Work

Recently, NAS has been achieved impressive results in many machine learning tasks. We categorize NAS research in three different classes according to their employed optimization methods: evolutionary algorithms (EA), reinforcement learning (RL) based approaches, and the efforts on optimizing CNN for disparity estimation. Plus, we review papers using multiple-objective optimization and handcrafted resource-aware architectures.

A. Reinforcement Learning Based Methods

RL-based NAS is composed of two basic components, a controller and the REINFORCE method. The controller which is usually based on Long short-term memory (LSTM), tries to generate the model descriptions of neural networks, e.g., kernel stride, kernel size, Batch Normalization. The REINFORCE method is responsible for updating controller weights by giving reward, e.g., accuracy, to the sampled architectures. [14] proposed a breakthrough method that uses a recurrent neural network (RNN) to design layer-wise architecture options trained by RL to maximize the accuracy of the generated architectures over the CIFAR-10 dataset. Later works follow a similar pipeline. NASNet [15] uses a modular exploration space to reduce the expensive exploration cost of the large datasets, e.g. ImageNet. NASNet first learns a basic architectural cell for a small-scale dataset, then it transfers it to a large-scale dataset by stacking together multiple cell copies. NASNet also introduced a new regularization technique which improve the generalization proficiency of the searched architectures. ENAS improves the low exploration speed of NAS by proposing a weight sharing scheme across child models during the NAS process [10]. [12] proposed an efficient NAS by reusing the current network weights in exploring new architectures for decreasing a large amount of the exploration cost. [16], [17] used Q-learning for updating the controller weights. The aforementioned methods only focus on accuracy improvement and the network resource utilization at the inference time is neglected. Recently, several proposed works try to improve accuracy while considering resource constraints as an exploration objective in order to design lightweight architectures. ProxylessNAS [6] directly explores the architectures for large-scale datasets while considering architecture latency. Plus, it proposes a path-level pruning scheme for decreasing high memory consumption of the conventional RL-based NAS algorithms. MNasNet [18] considers resource constraints in rewards explicitly. MNasNet leverages a novel factorized hierarchical exploration space to make a good balance between flexibility and exploration size. Most of the RL-based NAS still suffer from the prohibitive computational demand (e.g., [15] needs thousand GPU-days) which leads them to utilize proxy tasks, such as training

on a smaller dataset [14], [19], or training just for a few epochs [20]. However, compared to DenseDisp, the requiring exploration cost of RL-based NAS is unimaginable even in dreams (DenseDisp *directly* explores a architecture space with the minimum size of (20^{10}) in less than 2 GPU days). AutoDispNet [21] is the contemporary work to DenseDisp architecture. AutoDispNet is an efficient NAS for large-scale encoder-decoder architecture. To the best of our knowledge, AutoDispNet is the only NAS method optimizing disparity estimation. DenseDisp provides faster search compared to AutoDispNet while exploring a bigger space with conflicting objectives (see Section VI-A).

B. Evolutionary Methods

EA have a longer record in optimizing neural architecture [22]. In general, the best set of architectures is extracted in each iteration by using a "mutation" operation over a population of candidates. Several works rely on EA including [23], [24], [3], [2], [25], [26], [27]. In addition, many works utilize a multi-objective exploration considering resource budget and classification accuracy [27], [26], [3], [2]. Unfortunately, EA-based methods need enormous high-performance computing clusters (hundreds of GPU days), therefore, its usage is prohibited in poor budget circumstances. Moreover, they limit their usage to small learning tasks. Although DenseDisp is classified as an EA-based method, it solves existing weaknesses by proposing a simple single solution based optimization. In addition, DenseDisp addresses the impact of exploration objective on the quality of results (see Section VI).

C. Handcrafted Resource-Aware Models

Many handcrafted architectures have been proposed in order to obtain high accurate results on resource constraint devices. [28], [29] proposed IGCV models that generalize interleaved group convolutions to improve the sparsity of the structured kernels with the same model complexities. MobileNet [30] proposes inverted residual block to provide a trade-off between computational costs and accuracy. ShuffleNet [31] employs depth-wise convolution to reduce the model size while providing comparable accuracy level. [32] proposes CondenseNet, a novel combination of dense connectivity with learnable group convolution for improving the computational efficiency of DenseNet [33]. Although above architectures provide significant results, their design process is highly time-consuming and needs expertise. In addition, tweaking the hyper-parameters of these architectures, e.g., learning rate and weight decay, requires huge trial-and-errors. In comparison, DenseDisp is an automatic NAS which designs network architectures in a reasonable time without requiring any prior knowledge such as using optimized blocks extracted from small-scale tasks.

III. Exploration Space

A. Siamese Network Architecture

In general, Siamese neural network is a learning model that instead of directly classifying an image, tries to learn the similarity of two input images. It represents the stronger dissimilarity with distance a target as well. Siamese neural networks consist of two exact sub-networks ($N_W(X)$) which are parameterized by W. Siamese sub-networks share identical weights to measure the similarity between two input data (X_1, X_2). The goal of network training is to find W values in which the Euclidean distance $d(X_1, X_2) = |N_W(X_1) - N_W(X_2)|$ becomes minimized for the similar input data. The Siamese neural network is selected as flexible baseline architecture for disparity estimation where each sub-network processes the left or the right input image data (Fig. 2). [34] also utilized Siamese network for stereo vision task. However, instead of using a hand-crafted design, we leveraged NAS to find efficient Siamese sub-networks ($N_W(X)$) in terms of classification accuracy and the hardware resource budget represented by total number of network FLOPS. The Siamese architecture uses a simple inner product layer as the last layer to join the two network branches ($N_W(X_1), N_W(X_2)$) leading to decrease computation time. In this paper, we assume the image pairs are rectified, thus, the epipolar lines are aligned with the horizontal image axis. For training the architecture, [34] uses a small left image patches that were sequentially extracted from a set of pixels in which ground truth is available for them. While the size of the left image patch is equal to the network's receptive field, a bigger patch is used for the right image by expanding possible disparities and the size of the receptive field. These two patches are passed as input to the network for computing the score of each disparity range (typically 128 or 256). The Softmax activation function is then applied to the output of inner product layer over all possible disparities.

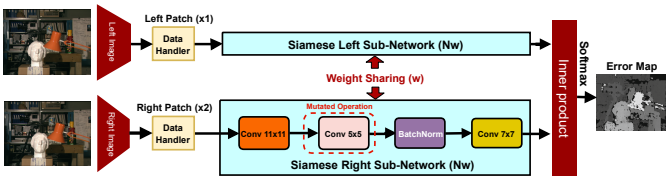

Fig. 2: A general overview of the Siamese neural architecture. $N_W(X)$ is the changing part of the network during the exploration.

B. Representation of CNN Exploration Space

We represent the exploration space by using a single directed list as the genotype where each node in the list acts the basic operations of the network. Fig. 3 illustrates a generic example of the genotype coded by a list of operations, named *List A*. Let us assume *List A* has N columns. In this representation, genotype is a flexible-length representation, where the size of N varies during the network evolution due to employing convolutional layers with the Valid padding and passive nodes. In this representation, passive nodes are defined as the operational nodes that are not involved in the processing flow (*Op#2* in Fig. 3).

Inspired from [2], We defined twenty one different operations including batch normalization, and all the combinations of Convolution_2D specified in Table I. We cannot use Pooling

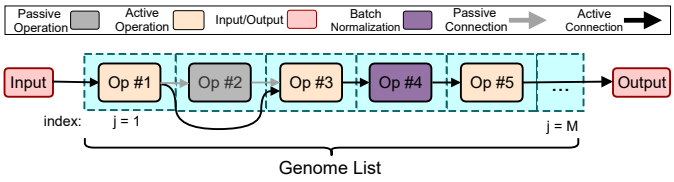

Fig. 3: A generic example of a genotype representing a CNN architecture in the search space.

layer because the size of feature maps in each dimensions should be odd. Here, we briefly present the procedure of constructing an architecture by stacking the network operations. First, we fill *List A* with random operations selected from Table I. In our experiments, the initial size of N is 10 columns. Afterwards, we connect the j^{th} column's node to the $(j-1)^{th}$ column's node in the list. Then, we connect the output feature maps received from the last node in the *List A* (*Nod#5* in Fig. 3) to the output node. The passive nodes are not involved in this procedure. DenseDisp supports two different mutation operations including: ① *swapping operations of two random nodes* and ② *replacing the operation of an active node with a valid operation shown in the Table I*. The size of exploration space depends on the length of *List A* in the genotype. The length of *List A* could be varied after mutation since DenseDisp supports convolution_2D nodes with *Valid padding*. Therefore, calculating the exact size of exploration is impossible. However, the minimum size of exploration space is 20^{10} if the length of the lists does not change when we have only convolution_2D operations with the *Same padding*. Obviously, the size of exploration space is much bigger.

TABLE I: The Specification of Node Operations.

Node Operation	Parameter	Value
Convolution_2D	padding	{Same, Valid}
	filter size	{32, 64}
	kernel size	$\{3\times3, 5\times5, 7\times7, 9\times9, 11\times11\}$
Batch Normalization		-

IV. Exploration

NAS is an NP-hard problem with an exponential time complexity. Thus, there is no polynomial optimization method to find the exact solution in a reasonable time. Exhaustive exploration methods for finding global optimum is unfeasible even for small exploration spaces. For example, [35] requires around 334 GPU days to exhaustively explore an exploration space with only 8000 points. Besides the complexity of the exploration space, evaluation of each architecture is time-consuming due to the huge training time. DenseDisp utilizes a meta-heuristic exploration in order to deal with the complexity of the NAS problem.

Algorithm 1: Pseudo-Code of the DenseDisp Exploration

Input: Training dataset, training and exploration configurations listed in Table III, exploration space configuration: *List A*, and *valid node operations specified in Table I*.
Result: Specification of an optimized Siamese architecture. DenseDisp Exploration (Input)
j := random initial architecture;
Initialize Boltzmann's constant k, reduction factor c, and temperature T
$T_{\text{Factor}} := -log(T_{\text{Max}}/T_{\text{Min}})$;
$Step := 0$;
while termination criterion is not satisfied **do**
  $Step$ += 1;
  j' := Mutate(j);
  if $Energy(j') < Energy(j)$ **then**
    $j := j'$;
  else
    random $r(0,1)$
    $\Delta := Energy(j') - Energy(j)$;
    if $r < exp(-\Delta/(k \times T))$ **then**
      $j := j'$;
    end if
  end if
  $T := T_{\text{Max}} \times exp(T_{\text{Factor}} \times (Step/Step_{\text{Total}}))$;
end while
return j

Exploration Method. In this paper, we used multi-objective Simulated Annealing (SA) as exploration method [36]. SA is a probabilistic single solution based optimization. SA is a fast optimization compared to population-based evolutionary optimizations such as genetic algorithm, and particle swarm optimization. SA works based on emulating the cooling process of a solid to eventually find a low-energy state. Algorithm 1 presents the pseudo-code of the DenseDisp exploration.

SA iteratively explores for a solution with fewer exploration objective value. Similar to the fitness function in genetic algorithm, exploration objective describes the optimality of a solution. If a reduction in exploration objective is found, the current solution is replaced with the new generated neighbour, otherwise the current solution is maintained. To avoid becoming trapped in a local optimum, SA sometimes accepts a bad solution which increases the value of exploration objective. The acceptance or rejection of a bad solution is dependent on a sequence of random numbers with a controlled probability. The acceptance probability is set to $exp(-\Delta/(k \times T))$ where T is the controlling parameter. T is the temperature inspired by the physical annealing process which is decreased logarithmic based on the predefined maximum and minimum temperatures (T_{Max} and T_{Min}). Thus, SA starts with a high value of T (T_{Max}) for preventing being prematurely trapped in a local optimum. Most uphill moves will be rejected by approaching T toward

T_{Min}. SA proceeds until no further improvements can be found or it will be terminated after a certain amount of iterations. Although SA theoretically may fail to find an optimal solution for a given budget, the experimental results demonstrate that it succeeds to find a near-optimal solution in a reasonable time (end-to-end 2 GPU days).

We use Equation 1 and Equation 2 to calculate the exploration objective. Normalizing the domain of multiple objectives is the main difficulty for designing the exploration objective. In Equation 1, the *Size_Factor* parameter is a proxy on the network size, where higher *Size_Factor* can be inferred from dense architecture with few FLOPS. *Max_FLOPS* is the total number of FLOPS for the largest possible designed architecture which is dependent to maximum the size of genotype. In this paper, we consider *Max_FLOPS* equal to 18.7×10^6 with assuming genotype with 10 columns. In Equation 2, The *Energy* parameter is the exploration objective designed to be in the range of 0 to 1. The α parameter is a practical coefficient which determines the intensity of compression. DenseDisp discovers accurate-oriented architectures for large α values, while more dense architectures will be designed by decreasing the value of the α parameter. In the rest of paper, the terms "Energy" and "exploration objective" are used interchangeably.

The convergency speed of our proposed method is highly dependent on the proper selection of the α parameter. Therefore, we consider four different scenarios with different α values to assess the impact of tweaking the exploration objective (Energy). Table II presents these four different scenarios considered in the experiments. Scenario ❶ stands for a balanced exploration when dense and accurate architectures have equal selection chance. In scenario ❷, DenseDisp tips the balance slightly in favor of accurate architectures. Unlike scenario ❷, DenseDisp moves toward discovering more dense architectures in scenario ❸. In Scenario ❹, DenseDisp gives an award to the architectures with accuracy more than 0.5 in order to improve the convergence by give more chance of success to accurate architectures.

$$Size\_Factor = \frac{Max\_FLOPS - FLOPS}{Max\_FLOPS} \quad (1)$$

$$Energy = (\alpha \times Accuracy) + ((1-\alpha) \times (1 - Size\_Factor)) \quad (2)$$

TABLE II: Specification of scenarios considered in the experiments with different α values to tweak the exploration objective.

Scenario	Value
❶	$\alpha = 0.50$
❷	$\alpha = 0.56$
❸	$\alpha = 0.44$
❹	$\alpha = 0.44$ if $Accuracy > 0.5$ then $Energy += 0.1$

V. Experimental Setup

KITTI-2015 [37] is a dataset of real world images that consist of 200 training and 200 test stereo pairs with the dimensions of height 376 pixels and width 1240 pixels. Compared to gray-scale images of KITTI-2012, KITTI-2015 directly processes on the RGB data. In this paper, we used the KITTI-2015 dataset for the the evaluations. Inspired by [3], we define the *Lookahead_Epoch* as the number of required training epochs for partially evaluating each architecture during the exploration step. Using partial evaluation for predicting the superiority of each network, trained by #*Lookahead_Epoch*, can highly accelerate the procedure of evaluating architectures. In this paper, we consider *Lookahead_Epoch* equal to 500 epochs since we can have 82% of maximum loss reduction after 500 epochs as shown in Fig. 4 (the loss reduction for the best model during the full training is illustrated in Fig. 4). Table III summarizes the setup of experiments utilized for the full training and exploration steps. Finally, we stop the exploration procedure if it cannot find any energy reduction after 10 iterations.

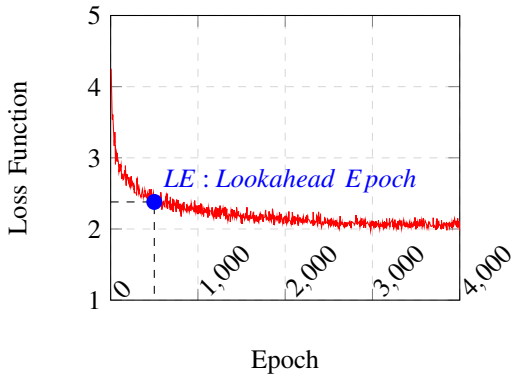

Fig. 4: The proceeding of the loss function during the full training of the final optimized DenseDisp architecture (Table IV).

VI. Evaluation

In this section, we evaluate the performance of DenseDisp on the KITTI-2015 dataset, and compare it with state-of-the-art solutions. *Disparity Estimation Performance*, *Inference/Exploration Speed*, *Analyzing Exploration Scenarios*, *Exploration Convergency*, and *Mutation Pattern of Dominant Node Operations* are recorded as the evaluation metrics over different exploration scenarios.

A. Disparity Estimation Performance

Table IV shows the accuracy of our DenseDisp architecture compared to existing networks. We select the best accuracy-FLOPS trade-off for the full training generated by the fourth scenario. Our DenseDisp architecture achieves 90.03% accuracy with 0.56 second inference time and 1.03M parameters/1.56M multiply-adds (FLOPS), achieving a higher accuracy-FLOPS trade-off for resource constraint disparity

TABLE III: The setup configuration of the training and exploration steps.

Full-Training Parameters	**Value**
Activation Function	Relu
# Epochs	4000
Batch Size	128
Disparity Range	128
Optimizer	AdaGrad
Learning Rate (lr)	0.01 and if (epoch$\geqslant$2400) then lr = lr/5 after each 800 epochs
Exploration Parameters	**Value**
# Lookahead_Epoch	500
Batch Size	8
Total Steps ($\text{Step}_{\text{Total}}$)	50000
Boltzmann's Constant k	$1.3807 \times 10^{-23} J \cdot K^{-1}$
Max Temperature (T_{Max})	25000
Min Temperature (T_{Min})	2.5
Hardware Configuration	**Value**
GPU	Nvidia GTX 2080
GPU Memory	11 GB
System Memory	64 GB

estimation. We observed that the median filter can significantly refine the intensity of the output disparity map (providing 2% accuracy enhancement). Median filter is an effective anon-linear digital filtering method that can be leveraged for reducing impulsive noises without smoothing the image edges [39]. In this paper, we applied a simple 2D median filter with a filter size of 15×15 pixels on the output of DenseDisp (Table IV).

DenseDisp provides $3.2\times$/$3\times$ faster inference time compared to GA-Net-deep as the most accurate network, while loses 8.16%/6.09% accuracy without/with considering median filter, respectively. In comparison with *AutoDispNet-BOHB-C* as an automatically designed architecture, DenseDisp has $39.91\times$ more network compression rate, while loses 7.79%/5.81% accuracy without/with considering median filter, respectively. It should be considered that unlike *AutoDispNet-BOHB-C*, we did not use any post optimization on the model hyper-parameters to improve the accuracy of the designed architecture. In terms of exploration time, DenseDisp runs $24\times$ faster optimization time (exploration + full training) which significantly outperforms *AutoDispNet* framework.

B. Analyzing Exploration Scenarios

Fig. 6 pictures the error-FLOPS trade-off between improved architectures (solutions with successful mutation) over different exploration objective scenarios. Scenario ❶ provides better balance between FLOPS and Error compared to a accurate-oriented exploration (Scenario ❷) and dense-oriented exploration (Scenario ❸). Scenario ❷ presents more accurate results with more FLOPS compared to Scenario ❸. Scenario ❹ ($\alpha = 0.44 + Award$) outperforms compared to other exploration scenarios due to eliminating weak solutions. In addition, Scenario ❹ fins more improved architectures over exploration time by escaping from local optima. The architecture with the best trade-off selected for the final full training is indicated by a green circle in Fig. 6.

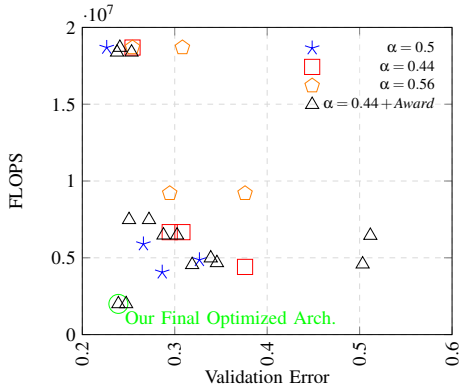

Fig. 6: FLOPS-error trade-off over different exploration scenarios. The green circle indicates the best designed architecture by DenseDisp selected for final full training.

C. Exploration Convergency

Fig. 5 illustrates the proceeding of FLOPS, Accuracy, and Energy over exploration iterations for the improved architectures (solutions with successful mutation). According to Fig. 5.a, in all the scenarios we find solutions with continuous reduction in FLOPS. On the other hand, DenseDisp does not necessarily approach toward improving accuracy (Fig. 5.b). Fig. 5.c shows the convergency of the exploration objective (Energy) guaranteeing that DenseDisp is always approaching toward improved solutions by decreasing the Energy level of improved architectures.

D. Analyzing Mutation Pattern of the Dominant Node Operations

Fig. 7 presents the contribution of each node operations (scenario ❹) in the discovered solutions over proceeding of the exploration method. Obviously, the kernel with size 5×5 and the *Same* padding (*Same:5x5*) is the dominant operation in the first random initial solution (iteration=1). However, the kernel with size 3×3 and the *Valid* padding (*Valid:3x3*) shows promising results by occupying more than 60% of nodes after 180 iterations. It means that DenseDisp selects *Valid:3x3* 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.

E. Disparity Map Outputs

Fig. 8 shows the results of CNN-based disparity estimation produced by DenseDisp (with/without median filtering).

VII. Conclusion

Directly solving the NAS problem on large scale tasks such as KITTI-2015 is highly expensive, as each architecture takes days to converge. Although many existing approaches perform architecture exploration on smaller tasks such as CIFAR-10, using small proxy tasks is not efficient when the architecture size is taken into account [18]. In this paper, we first present a varying exploration space to provide a vast options for network selection. Then, we performed a direct architecture

TABLE IV: Comparing DenseDisp with other state-of-the-art methods on KITTI-2015 dataset.

Architecture	**Accuracy (%)** $D1-all$	**Params** $\times 10^6$	**FLOPS** $\times 10^6$	**Exploration Method**	**Compression Rate† (%)**	**Exploration Cost (GPU Days)**	**GPU Latency (Sec.)**	**H.W. Platform**
AutoDispNet-BOHB-C†[21]	97.82	37	61	RL	1	42	-	1x GTX 1080ti
Content-CNN [34]	95.46	7	2	Hand-Crafted	35.5	-	1	1x Titan Xp
GA-Net-deep [38]	98.19	-	-	Hand-Crafted	-	-	1.8	1x Tesla P40
DenseDisp (Ours)	90.03	1.03	1.56	Meta-heuristic	39.1	2	0.56	1x GTX 2080
DenseDisp + Median Filter (Ours)	92.01	1.03	1.56	-	39.1	2	0.6	1x GTX 2080

† The baseline for comparing the compressing rate. Our considerable results are in green cells.

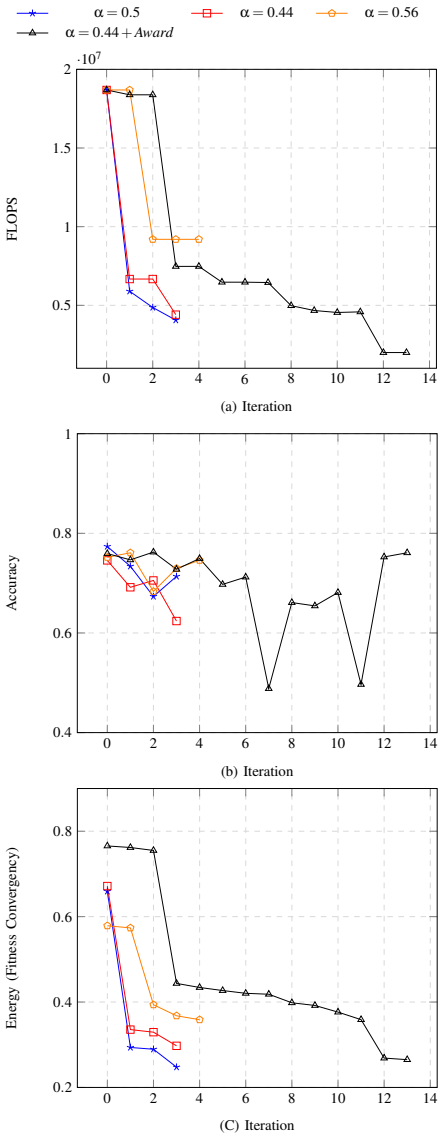

Fig. 5: Convergency of the (a) Accuracy, (b) FLOPS, and (c) Energy parameters.

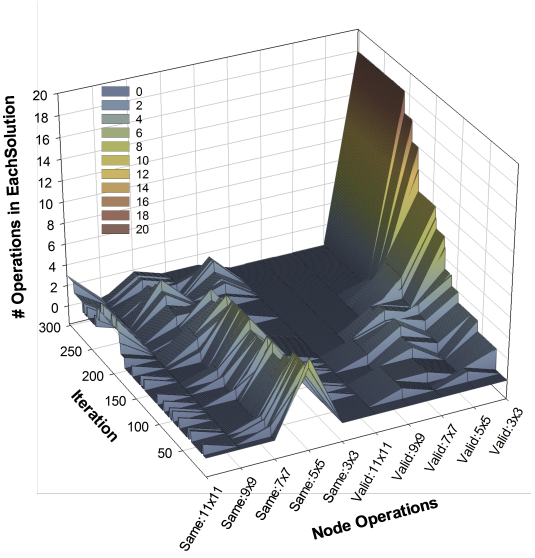

Fig. 7: Analyzing dominance pattern of each node operation (Table I) in the mutated architectures over proceeding the search iterations.

exploration since utilizing the pre-learnt cells in exploring new architectures for new task is not guaranteed to be efficient [6]. Third, we consider the network FLOPS along with accuracy as the exploration objectives to discover architectures with balanced FLOPS-accuracy trade-off for resource-constraint hardware.

VIII. Acknowledgement

This Paper is supported by Knowledge Foundation (KKS) and HiPEAC collaboration grant within DeepMaker project.

References

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

[2] 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.

[3] 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*, p. 102989, 2020.

[4] F. Hutter, L. Kotthoff, and J. Vanschoren, "Automated machine learning-methods, systems, challenges," 2019.

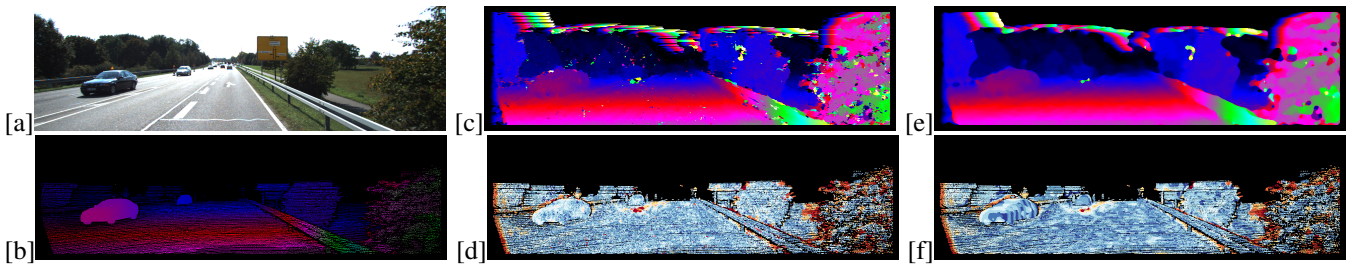

Fig. 8: (a) Input image. (b) Ground truth. (c) DenseDisp disparity map (d) Error (DenseDisp) (e) {DenseDisp + Median Filter} disparity map (f) Error ({DenseDisp + Median Filter})

[5] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search," in *Automated Machine Learning*. Springer, 2019, pp. 63–77.

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

[7] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[8] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," *arXiv preprint arXiv:1810.05749*, 2018.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[10] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[11] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[12] 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.

[13] R. Tang, W. Wang, Z. Tu, and J. Lin, "An experimental analysis of the power consumption of convolutional neural networks for keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5479–5483.

[14] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[15] 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.

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[17] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2423–2432.

[18] 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.

[19] Y. Zhou, S. Ebrahimi, S. Ö. Arık, H. Yu, H. Liu, and G. Diamos, "Resource-efficient neural architect," *arXiv preprint arXiv:1806.07912*, 2018.

[20] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[21] T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox, "Autodispnet: Improving disparity estimation with automl," *arXiv preprint arXiv:1905.07443*, 2019.

[22] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[23] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2902–2911.

[24] 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.

[25] 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.

[26] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: a multi-objective genetic algorithm for neural architecture search," *arXiv preprint arXiv:1810.03522*, 2018.

[27] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[28] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi, "Interleaved structured sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8847–8856.

[29] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang, "Interleaved group convolutions," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4373–4382.

[30] A. Howard, A. Zhmoginov, L.-C. Chen, M. Sandler, and M. Zhu, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," 2018.

[31] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[32] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761.

[33] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[34] 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.

[35] 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.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[37] 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.

[38] 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.

[39] D. A. Florencio and R. W. Schafer, "Decision-based median filter using local signal statistics," in *Visual Communications and Image Processing'94*, vol. 2308. International Society for Optics and Photonics, 1994, pp. 268–275.