2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-7281-8470-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICMLA51294.2020.00022

Neural Architecture Search for Automotive Grid Fusion Networks Under Embedded Hardware Constraints

Gabor Balazs
Technische Universität München
Munich, Germany
gabor.balazs@tum.de

Walter Stechele
Technische Universität München
Munich, Germany
walter.stechele@tum.de

***Abstract*—The goal of automotive sensor fusion is to generate an environmental model with low-cost, mass-produced sensors—typically camera and radar. The central fusion controller needs sophisticated post-processing algorithms to achieve this goal despite extensive pre-processing on the sensor side. Recent advances show the capabilities of deep convolutional auto-encoders for sensor fusion but are still limited to high-performance hardware. Limited memory and processing capabilities of embedded devices lead to the multi-objective optimization problem addressed in this work. The proposed evolutionary neural architecture search is configured to find novel sensor fusion network architectures optimized for embedded hardware. The discovered encoding and decoding cells improve fusion performance compared to that of corresponding cells from literature. These lightweight network architectures allow the deployment on constrained embedded devices, thus leverage environmental perception of level 2/2+ automated cars.**

***Index Terms*—neural architecture search; sensor fusion; convolutional neural network;**

I. Introduction

Mobile robots like automated vehicles require an accurate representation of their environment for navigation tasks. In order to generate such an abstract environmental model robustly, multiple types of sensors are employed. By fusing the multimodal data, shortcomings of each sensor type are overcome and synergies are exploited.

Advanced driver assistance systems rely on accurate environmental models that are robust to weather conditions, and that are based on redundant information in case of single sensor failures. A typical sensor setup used in mass-market cars is a combination of smart camera and radar sensors. The two modalities cover information that complement each other in their lateral/radial accuracies and their ability for dynamic/static object detection and classification. These distributed sensors incorporate pre-processing steps on sensor-side, thus dispose only feature-level data to the central fusion ECU.[1] Because of the different nature of sensor data, this work jointly integrates camera and radar information into a common representation, namely the occupancy grid (OG).

[1]As seen in the system architecture of the camera and radar system used in Tesla's Autopilot v1, low bandwidth communication does not allow raw data streams—feature-level data only.

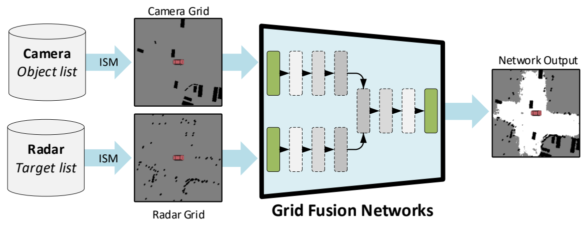


Fig. 1. System overview of grid fusion networks: Feature-level data from smart sensors are translated into OGs via inverse sensor models (ISM). Grid fusion networks process the input grids and output a fused representation.

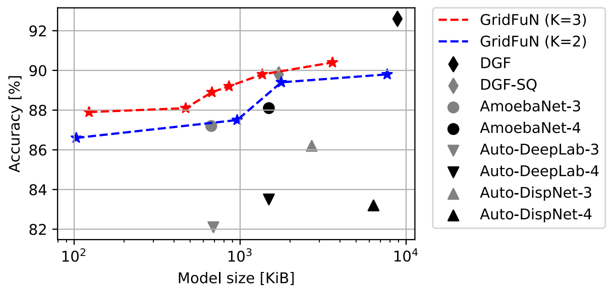


Fig. 2. The trade-off between model size and fusion accuracy for different architecture settings ($K = 2$, $K = 3$). The proposed model families outperform cells from *Auto-DeepLab* [1] and *Auto-DispNet* [2] with $\delta \in \{3, 4\}$.

Classical grid fusion algorithms like Bayesian or Dempster-Shafer fusion work well on raw Lidar data but have problems at generating dense environmental models instantaneously. As OGs are two-dimensional image-like matrices, they can be processed by convolutional neural networks (CNNs) naturally. In consequence, the grid fusion task can be done by using CNNs at the cost of additional computational burden.

Automotive microcontrollers have hard constraints in terms of computational resources and available memory. In order to run CNNs on them, the networks have to be compressed and optimized for inference. One way is to search for an architecture manually that addresses both high-quality fusion and low computational complexity. Recent advances in neural architecture search (NAS) show that the automated approach outperforms manually designing networks. The novel approaches create network architectures that are superior to handcrafted ones, especially when approaching multi-objective optimization problems.

In this work, the problem of the fusion of feature-level sensor data in automotive is approached with an evolutionary NAS. The network structures discovered by the proposed framework outperform corresponding blocks from *Auto-DeepLab*, *Amoeba-Net* and *AutoDispNet* in terms of fusion quality and model size.

The key contributions of this work can be summarized as follows:

- A NAS framework that is configurable for arbitrary input branches and image dimensions. This flexible design allows to search for network architectures different from standard classification or segmentation networks, such as the proposed grid fusion networks.
- An extended search space for operations within a cell with an adaptive strategy for reshaping tensors in between cells of the macro-architecture.
- A family of grid fusion networks optimized for embedded implementation (`GridFuN`). The model sizes of the networks range from 90 KiB to 4 MiB. The fusion performance increases proportional to the memory footprint.

II. Related Work

a) Occupancy Grid Processing: The concept of partitioning surronding space into grid cells with properties like occupancy probability, namely the occupancy grid, was introduced by [3]. A generalized framework for handling grids is published in [4], where modeling, alignment, and transformations of grids for fusion and mapping are defined.

Several works are about using CNNs to process occupancy grids. [5] uses networks to detect and locate objects in OGs, whereas [6] classifies pixels of occupancy grids into several classes, such as *buildings* or *moving*. The usage of CNNs for fusing two OGs was investigated in [7] with an auto-encoder architecture similar to U-Net [8] or Chi-Net [9].

b) NAS for Segmentation Tasks: Whereas image classification has significantly improved through automated machine learning algorithms such as neural architecture search, image segmentation got less attention. Using NAS to find optimal architectures for image segmentation is mostly applied for medical image analysis. [10] propose *SCNAS* for 3D medical image segmentation with the architecture similar to U-Net, but with searchable *encoding*, *encoding-normal*, *decoding*, and *decoding-normal* cells. *NAS-Unet* [11] concentrates the search on only two lightweight cell types (*DownSC*, *UpSC*) and replace standard skip connections with `cweight` operations known from *Squeeze-and-Excitation* networks [12]. In *Auto-DeepLab*, cell-level and network-level architectures for general image segmentation are jointly searched [1]. Disparity estimation is improved by *AutoDispNet* with an auto-encoder structure of *encoding*, *decoding*, and *normal* cells [2].

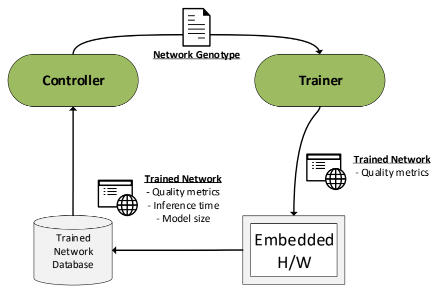


Fig. 3. The principle of neural architecture search.

Different search strategies are used for NAS. Most prominent methods are reinforcement learning [13]–[15], and evolutionary algorithms—employed for *AmoebaNet* [17] and other works [16], [18], [19].

c) Optimization for Mobile: Recent works on NAS focus on awareness to platform-specific memory footprint and accelerator topology. Aiming for mobile deployment, [20] propose *MnasNet* and conduct a multi-objective optimization in order to find multiple Pareto-optimal network architectures regarding accuracy and inference time. Similar to *MnasNet*, [21] propose *MobileNetV3* for mobile devices, but focus also on semantic segmentation.

III. Evolutionary Neural Architecture Search

Based on the comparative research of [17], an evolutionary NAS was chosen. In the genetic algorithm each network architecture is described by a genotype that defines the computational graph and all the necessary hyper-parameters. The cell-wise approach with a fixed macro-architecture known from [13] was adapted and extended to a framework that can be configured to arbitrary network macro-architectures. In this work, the NAS is configured to search for grid fusion networks `GridFuN` with multiple input branches to an encoder–decoder cell structure and one output image.

The NAS of this work is structured into a controller that coordinates network creation and a trainer that evaluates network performances (Fig. 3). The network database, from which the controller coordinates the evolution process, stores only the network genotype and its evaluation metrics.

In Algorithm 1, the procedure of one NAS experiment is described. This method searches for networks that are optimized to process data coming from dataset D on a given embedded hardware H.

Algorithm 1 The framework of `GridFuN`

```
Input: $e$ epochs, $p$ population size, $n$ nodes per cell, $f$ initial filters, dataset $D$, embedded hardware $H$.
Output: GridFuN architecture family
$P$ = initialize ($p$, $n$, $f$)
for $i = 1$ to $e$ do
    $P'$ = sample ($s$, $P$)
    $P'$ = mutate ($P'$)
    $P'$ = train ($P'$, $D$)
    $P'$ = evaluate ($P'$, $H$)
    $P$ = $P \cup P'$
end for
```

A. *Grid Fusion Networks*

The networks in this work process occupancy grids (OGs). They describe an area around the car with discrete, spatial cells containing occupancy information. The grids have the dimensions $H \times W \times F$, where H and W denote the width and height, and F the number of feature channels of the grid. Here, $F = 3$ (c_1: free, c_2: unknown, c_3: occupied). As the OGs are two-dimensional matrices like images, they can easily be processed with convolutional networks.

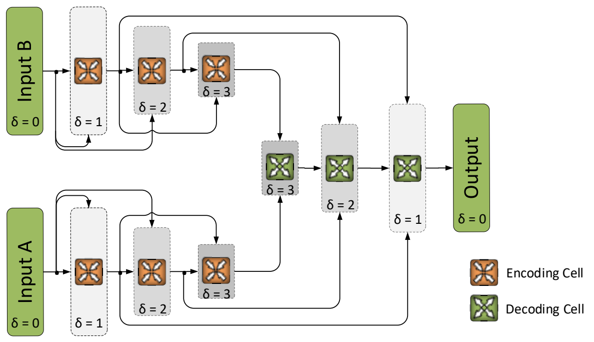


Fig. 4. Macro-architecture of a grid fusion network. The encoding depth is variable but set to $\delta = 3$ for visualization.

B. *System Design—Macro-architecture*

The evolutionary algorithm makes use of a predefined macro-architectural skeleton that describes the arrangement of cells C within each network. The genotype of each network can be configured to a fixed macro-architecture by describing the hard-wired cell-wise connections and bypasses. A cell can be configured to have an arbitrary number of inputs. As the desired macro-architecture for processing OGs is a convolutional auto-encoder, two separate cell types are searched—encoding C_E and decoding cells C_D.

The system was designed to perform an architecture search for a grid fusion problem with multiple occupancy grids (e.g. OG_A, OG_B)[2] as inputs and one occupancy grid as the output. Fig. 4 shows the configuration of the connections between the cells. The macro-architecture is configured to be similar to U-Net [8] but with two input images. Each input branch is a stack of δ encoding cells $C_E^0, ..., C_E^\delta$. In each encoding cell, the tensor $W \times H \times F$ is transformed to the output tensor $W/2 \times H/2 \times 2F$.

After δ encoding cells, the fusion of the branches A, B starts by combining the outputs of all cells of the same depth δ. The first decoding cell C_D^δ has two inputs—the last encoding cells of each input branch—and all the following decoding cells $C_D^{\delta-1}, ..., C_D^0$ have a third input of the preceding decoding cell. The decoding cells C_D take input tensors of dimension $W \times H \times F$ as an input and transform them to one output tensor $2W \times 2H \times F/2$.

The normal and residual connections are arranged according to the setup in [10], where each encoding cell has an input from the two previous ones, except for the first encoding cell that is fed with the input layer alone.

C. *Method Details—Micro-architecture*

The actual search is performed on the micro-architecture level within the cells. Each cell C is described by a directed acyclic graph (DAG) with Knodes.

A node is a *Y-shaped* structure that maps two input tensors to one output tensor. Node N_i^δ is the i-th node in a cell of depth δ. It is described by a 5-tupel (I_1, I_2, O_1, O_2, O_+) with the two node input tensors $I_1, I_2 \in \mathcal{I}_i^\delta$, the two operations $O_1, O_2 \in \mathcal{O}$ that are applied to the corresponding inputs, and the combination operation $O_+ \in \mathcal{O}_+$ that combines the two tensors (Fig. 5a). The sets of possible operations $\mathcal{O}$ and combinations $\mathcal{O}_+$ are described in the search space paragraph below.

The node inputs are selected from the set of tensors $\mathcal{I}_i^l$ that is a union of the two previous cell outputs and the previous node outputs within the current cell $\mathcal{I}_i^\delta := \{C^{\delta-1}, C^{\delta-2}\} \cup \{N_j^\delta \mid j \in \mathbb{N} \wedge j < i\}$.

There are two cell types, one used for encoding branches (C_E) and one the decoding part (C_D). The encoding cells reduce image dimensions by 2 with strides $S = 2$ at the operations coming from the cell input tensors $C^{\delta-1}, C^{\delta-2}$. In decoding cells, the image dimensions are kept constant ($S = 1$), only after the combination operation O_+, the image is upsampled with an upsampling operation $O_{\text{up}} \in \mathcal{O}_{\text{up}}$ (Fig. 5b).

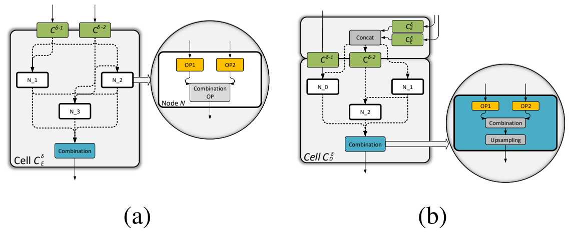


Fig. 5. Generic micro-architectures of (a) encoding and (b) decoding cells. Each cell has two inputs and one output that is computed over K nodes (here $K = 3$ plus one O_+). The generic decoding cells concatenate bypassed input tensors with O_+^{cat} before addressing the same cell architecture like encoding cells.

[2]The number of input branches is set to two throughout this work, but the NAS can be configured for processing more inputs.

a) Search Space: The search space $\mathcal{O}$ is inspired by the work of [13], but is extended to $|\mathcal{O}| = 13$ operations. The set of normal operations $\mathcal{O}$ consists of convolutions $O^{\texttt{conv}}$, depthwise separable convolutions $O^{\texttt{sep}}$, and dilated convolutions $O^{\texttt{dil}}$ with filter sizes of $\{1 \times 1, 3 \times 3, 5 \times 5\}$, skipped connections $O^{\texttt{skip}}$, max pooling 3×3 $O^{\max}$, and average pooling 3×3 $O^{\texttt{avg}}$. Additionally, also the *fireModule* known from *SqueezeNet* [22] is included in the search space as $O^{\texttt{fire}}$.

The set of combination operations $\mathcal{O}_{+}$ consists of the element-wise addition $O_{+}^{\texttt{add}} = \sum_i O_i$ and the concatenation along the feature channels $O_{+}^{\texttt{cat}}$. The set of upsampling operations $\mathcal{O}_{\texttt{up}}$ consists of transposed convolutions $O_{\texttt{up}}^{\texttt{t\_conv}}$ of different filter size and the bilinear interpolation $O_{\texttt{up}}^{\texttt{interp}}$.

b) Reshaping Strategies: At some configurations of cells, the dimensions of the inputs to a combination operation $\mathcal{O}_{+}$ do not fit, but subsequent combination operations require equal dimensions to work correctly. Thus, adaptive reshaping is necessary to build a valid network model.

Reshaping involves image dimension adjustment and feature channel adaptation. Image dimensions have to be reshaped, whenever an encoding cell C_E^{δ} is bypassed and the bypassing tensor $C_E^{\delta-1}$ has different dimensions compared to the output of this encoding cell. The correction follows through additional adjustment operations $O_{\texttt{adj}} \in \mathcal{O}_{\texttt{adj}}$. The set of $\mathcal{O}_{\texttt{adj}}$ consists of convolution or pooling layers with strides $S = 2$ for downsampling.

A change in the number of feature channels F is needed, whenever operations are used that themselves do not change the number of feature channels, such as normal pooling OPs. The F correction can be done with simple 1×1 convolutions or by changing the combination operation from $O_{+}^{\texttt{add}}$ to a $O_{+}^{\texttt{cat}}$ along the last dimension.

Following the work of [23], the two strategies for image size adjustment are explored during the search. Using convolutions with strides for image dimension reduction early in the network significantly reduces the amount of computations needed in subsequent layers and has limited loss of information. In contrast, using pooling layers is beneficial in reshaping skipped connections, where no additional weights have to be learned, thus gradient propagation is eased.

D. Genetic Diversity

The population is initialized with each network having randomly chosen DAGs as cell descriptions. After training the first generation of the population P, the best architectures are determined according to the fitness function (Sec. III-E). From these chosen architectures, offsprings are created with slightly mutated properties, and the next generation is created (Algorithm 1). To maintain genetic diversity of the current population, the following four different mutations are implemented:

a) Operation Mutation—M_{op}: Within a random node in a random cell type, one of the operations O_i is selected. After the mutation M_{op} the operation becomes $O_i' \in \mathcal{O} \backslash \{O_i\}$.

Algorithm 2 Choice of Mutation

```
Input: GridFuN architecture $x$
Output: Mutated GridFuN architecture $x$
$r$ = randomInteger(1, 3)
for $i = 1$ to $r$ do
    $m$ = randomInteger(1, 100)
    if $m \leq 30$ then
        $x = M_{\texttt{op}}(x)$
    else if $30 < m \leq 60$ then
        $x = M_{\texttt{con}}(x)$
    else if $60 < m \leq 90$ then
        $x = M_{\texttt{fmap}}(x)$
    else if $m > 90$ then
        $x = M_{\texttt{init}}(x)$
    end if
end for
```

b) Connection Mutation—M_{con}: If this mutation is called, one of the inputs $(I_1, I_2) \in \mathcal{I}_i^{\delta}$ to a random node i will be changed, so that (I_1, I_2) is becoming either (I_1', I_2) or (I_1, I_2') with $I_1 \neq I_1'$ and $I_2 \neq I_2'$, respectively.

c) Number of Filter Maps Mutation—M_{fmap}: Also the number of initial feature channels is searchable as a hyper-parameter of the grid fusion networks. This mutation allows the NAS to find an optimal number of filters F per convolutional layer. With every call of this mutation, F increases or decreases by 4. The minimal number of feature channels is $F_{\texttt{min}} = 4$.

d) Cell Re-initialization—M_{init}: With the call of $M_{\texttt{init}}$, a cell type is re-initialized in order to avoid local minima of network architectures. The random cell type is chosen and reset to a random configuration—ignoring the configuration before.

Each offspring of a parent architecture is mutated according to Algorithm 2. Before the model is trained, it is checked whether this architecture configuration has already been evaluated and stored in the network database. This saves time because duplicate networks are not trained twice.

E. Multi-objective Optimization

In order to fit networks onto strictly constrained embedded hardware, optimizing the networks only for accuracy metrics is not enough. Model size has to be shrunk and network architecture chosen according to the capabilities of the hardware accelerators. The NAS boils down to a multi-objective optimization with specific constraints dependent of the embedded hardware at hand.

The grid fusion quality is evaluated with performance metrics for semantic segmentation tasks. Four quality metrics from the PASCAL VOC (pixel accuracy pAcc, mean accuracy mAcc, mean intersection over union mIoU and frequency weighted IoU fIoU) are measured after the training [24].

During the search process, the networks are ranked according to their fitness value. The networks with the highest fitness value form the current population and generate offspring networks. As a consequence of the aforementioned objectives, the fitness value is dependent on multiple factors that have to be balanced. The fitness function is a sum of six separate reward functions, each described in the following.

a) Quality Rewards: For given network x, the rewards are calculated as follows. To avoid bias towards any of the metrics, the reward for each has an upper bound $R_{\texttt{max}}$ that is defined by the upper bound for all metrics λ. The rewards for each metric μ are calculated as

$$R_{\mu}(x) \begin{cases} \frac{1}{1-\mu_x}, & \mu_x < \lambda, \\ \frac{1}{1-\lambda}, & \mu_x \leq \lambda. \end{cases} \tag{1}$$

The four metrics from PASCAL VOC are translated into the combined quality reward

$$R_q(x) = \sum_{\mu_x} R_{\mu}(x), \tag{2}$$

with $\mu_x \in \{\texttt{pAcc}(x), \texttt{mAcc}(x), \texttt{mIoU}(x), \texttt{fIoU}(x)\}$.

b) Inference Time: The averaged execution time on the embedded hardware t_x is used to calculate the inference time reward

$$R_t(x) = \begin{cases} \frac{t_{\texttt{min}}}{t_x} R_{\texttt{max}}, & t_x > t_{\texttt{min}} \\ R_{\texttt{max}}, & t_x \leq t_{\texttt{min}}, \end{cases} \tag{3}$$

with a lower boundary of inference time $t_{\texttt{min}}$ and the maximum reward $R_{\texttt{max}}$ for meeting the latency requirements of the task (e.g., $50\,Hz$ real-time requirements).

c) Model Size: The total memory footprint m_x—on-chip- and off-chip-memory combined—is taken into account in the model size reward

$$R_m(x) = \begin{cases} \frac{1}{m_x} R_{\texttt{max}}, & m_x > m_{\texttt{min}} \\ R_{\texttt{max}}, & m_x \leq m_{\texttt{min}}, \end{cases} \tag{4}$$

with $m_{\texttt{min}}$ as the minimal required memory footprint and $R_{\texttt{max}}$ as the maximum reward for meeting the memory requirements of the hardware.

d) Fitness Function: Given a proposed network model x, and the respective rewards $R_t(x)$, $R_m(x)$, and $R_q(x)$, let $f(x)$ denote the network's fitness value

$$f(x) = R_t(x) + R_m(x) + R_q(x). \tag{5}$$

With a given, hardware-dependent set of ($t_{\texttt{min}}$, $m_{\texttt{min}}$), the goal of the neural architecture search is to find multiple Pareto-optimal solutions. Thus, the multi-objective optimization is maximizing the fitness function

$$\max_{x} \quad f(x). \tag{6}$$

IV. Evaluation and Results

In this section, the training procedure and the resulting cell micro-architectures are described, and the performance of `GridFuN` architectures are compared to works from literature.

A. Network Training Environment

The macro-architecture during the experiments is set to an encoding depth $\delta = 4$, so that the lowest dimensions of the tensors are $H/16 \times W/16 \times 16F$. Two experiments are conducted, with cells having $K = 2$ and $K = 3$ intermediate nodes with initially 8 feature channels in convolutional layers.

a) Grid Fusion Dataset (`GFD`): As introduced in Section I (Fig. 1), the inputs to the fusion networks are pre-processed sensor data from camera and radar. These kind of data is extracted from the nuScenes dataset [25]. To mimic the feature-level nature of pre-processed data, the provided object annotations are taken as camera detections. These, and the radar data are transformed into occupancy grids by applying inverse sensor models. The ground truth (GT) is based on the free space estimate of the Lidar data. Each OG has the dimension $256 \times 256 \times 3$, where the three feature channels specify the occupancy states (s_1: free, s_2: unknown, s_3: occupied). With a cell resolution of $0.25\,m$, the total grid size is $64\,m \times 64\,m$. The ego vehicle is positioned in the center of each OG, and all sensor measurements used to generate the OGs are synchronized in time and space.

NuScenes incorporates 1,000 scenes of $20\,s$ length that, in total, form 44,760 frames in the dataset `GFD`, with each frame consisting of three OGs (radar, camera, and GT). The subset $\texttt{GFD}_{\texttt{light}} \subseteq \texttt{GFD}$ consists of 5,632 frames based on the first 100 scenes of *nuScenes*.

b) Latency Reduction: In order to reduce evaluation latency, the networks are partially trained on $\texttt{GFD}_{\texttt{light}}$ for 2 epochs. The best performing architectures of the search process are then retrained on the complete `GFD` to display their full performance. The full training cycle consists of 10 epochs in contrast to the reduced training setup during the search process. Empirically, the final performances of the networks are proportional to the performances of the partially trained networks. From these observations we can derive that $f(a') > f(b') \rightarrow f(a) > f(b)$ holds for fully trained networks a, b, and partially trained networks a', b'. This way, the return time of an average network evaluation is reduced down to about 10 minutes, instead of several hours.

c) Training Procedure: All experiments are conducted relying on Tensorflow 1.13.1 [26] with compiler version 5.4.0 and the training is performed on an Nvidia Titan V GPU under CUDA 10.0.130.

All grid fusion networks are trained on $\texttt{GFD}_{\texttt{light}}$ at a split of 80% training samples and 20% validation samples. The evaluation metrics are averaged over 10 batches with a batch size of 16. The learning rate α is decreasing from $\alpha_0 = 0.0005$ to zero according to a cosine decay (Eq. 7) with a warm-up 2.5% and a hold until 10% of the total iterations i_T, according to

$$\alpha(i) = \begin{cases} \alpha_0 \ \frac{i}{0.025\ i_T}, & i < 0.025\ i_T \\ \alpha_0, & 0.025\ i_T \leq i \leq 0.1\ i_T \\ \alpha_0\ \cos(\frac{2\pi}{i_T}\ i), & i > 0.1\ i_T, \end{cases} \tag{7}$$

where i is the current iteration.

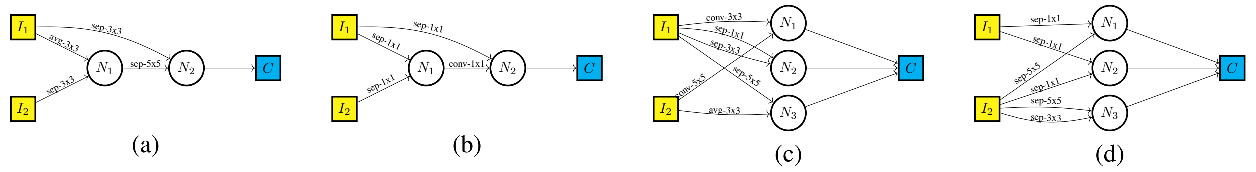


Fig. 6. Connection graphs of the best cell architectures with different number of intermediate nodes K. `GridFuN`-S ($K = 2$, $F = 16$): (a) Encoding cell C_E, (b) decoding cell C_D. `GridFuN`-L ($K = 3$, $F = 12$): (c) Encoding cell C_E, (d) decoding cell C_D.

The training during the search process of the NAS is done for 2 epochs. The full training for comparison is done for 10 epochs. The ADAM optimizer is used with default parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ to minimize the pixel-wise softmax classification loss.

The networks are trained in floating point 32-bit precision and quantized after training to 8-bit unsigned integers for deployment on the *EdgeTPU*. Also, not all operations could be performed on the *EdgeTPU*; the upsampling operations are done with nearest-neighbor interpolations instead of transposed convolutions. Once deployed on the embedded hardware, inference time is measured by averaging 100 inference runs.

Two experiments are conducted, one with the $K = 2$ nodes within a cell, and one with $K = 3$. The auto-encoder depth is chosen to be $\delta = 4$. The final performance results are received from networks that have been trained on the full dataset for 10 epochs. In the following, the resulting cell micro-architectures are described.

B. Discussion of Results

After the evaluation of 1,000 trained networks, the resulting Pareto-optimal cell architectures for each problem are extracted. Two exemplary architectures are depicted in Fig. 6.

Cell architectures from the relevant literature are compared to the proposed `GridFuN` architectures. To have a fair comparison, only cells from segmentation tasks, such as medical image segmentation are compared to our cells for the grid fusion task. The cells of *Auto-DispNet* [2], *Auto-DeepLab* [1], and *AmoebaNet* [17] are used for comparison.

In detail, the cells of these frameworks are plugged into the grid fusion macro-architecture: The encoding and decoding cell from *Auto-DispNet*, and the best cell architecture from *Auto-DeepLab*. Also, the reduction cell from the NAS-based classification network *AmoebaNet* [17] is employed as the encoding cell, and the decoding cell is taken from the `GridFuN`-L architecture.

All networks in the comparison are trained with the training conditions (dataset, training parameters, etc.) described in Sec. IV-A. Table I shows the performance metrics along with architectural parameters of the networks. Manually designed architectures from [7] show high performance, but large sizes compared to NAS-based approaches. The searched architectures from literature lack the optimization towards the embedded hardware. Thus, they have lower fitness values due to the large network sizes and inference times.

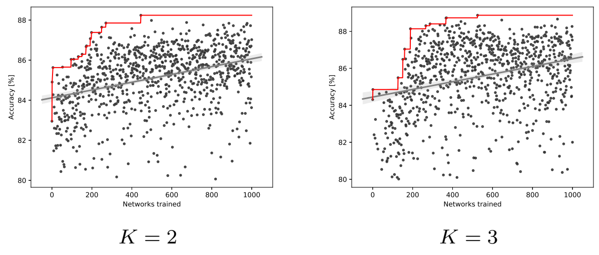


Fig. 7. Evolution of accuracies over the two experiments. Both experiments show that the NAS converges after about 500 trained networks. The gray line indicates the mean accuracy rising over time.

In Fig. 8 the Pareto-optimal network architecture performances are shown for two experiments ($K = 2$, $K = 3$). The red line shows the accuracies over memory sizes of the best performing networks during the search process. The blue stars mark the performance of the fully re-trained networks.

a) Bias Towards Certain Network Architectures: Because the search process is limited to partial training of the networks, architectures evolve that tend to train faster. This has the downside that architectures with more potential performance are discarded because they cannot reach their maximum performance during the short training time.

Also, networks that have errors during the embedded deployment are penalized in the rewards R_t and R_m, thus hindered from the further evolution, and finally excluded from comparisons. The deployment errors are based on compilation errors with *TensorFlow Lite converter* because of the model requirements for the *EdgeTPU*.

V. Conclusion

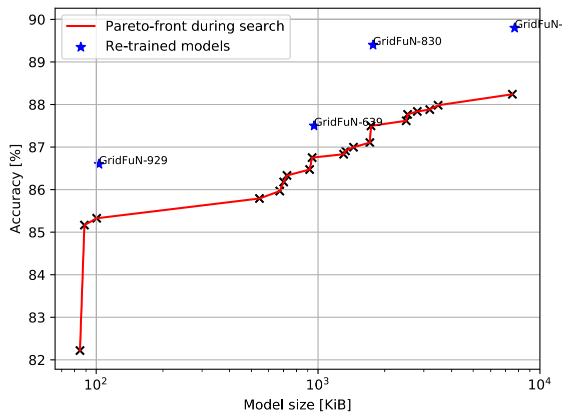
In this work, a NAS framework for arbitrary macro architectures is proposed that is configured and evaluated for sensor fusion networks. The backbone of the architectures is a convolutional auto-encoder structured into two types of searchable cells (encoding C_E and decoding cells C_D). The NAS finds the optimal structure of those cells under constraints of automotive microcontrollers with limited memory and compute capabilities. The standard search space from [13] is enlarged by dilated convolutions and fire modules. These operations have advanced capabilities to capture contextual information and reduced computational requirements, respectively.

The family of `GridFuN` architectures have a range of model sizes from 90 KiB to 4 MiB. The fusion performance increases proportional to the memory footprint. Empirical

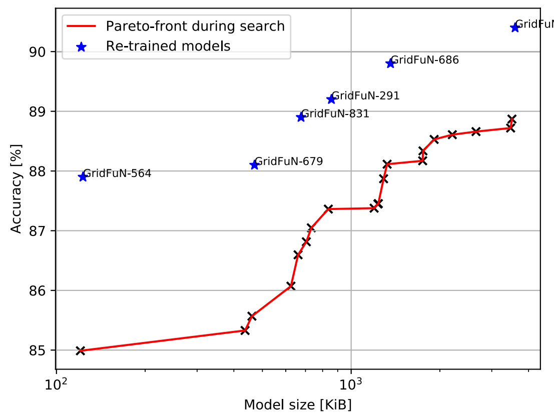
TABLE I

PERFORMANCES OF DIFFERENT CELLS IN THE GRID FUSION MACRO-ARCHITECTURE. (*) *Auto-DeepLab* [1] CELL PLACED AS C_E AND C_D. (**) *AmoebaNet-A* [17] REDUCTION CELL AS C_E AND BEST GRIDFUN DECODING CELL. (***) *Auto-DispNet* [2] ENCODING AND DECODING CELLS. [1,2] BEST PERFORMING NETWORKS FROM [7] AFTER FULL TRAINING. DGF USES UNSUPPORTED OPERATIONS THAT COULD NOT BE MAPPED TO THE *EdgeTPU*.

METHODS	K(C_E/C_D)	F	δ	NAS	ACC. [%]	MIOU [%]	SIZE [MIB]	LATENCY [MS]	FITNESS
DGF[1]	N/A	16	5		**92.6**	**85.2**	8.87	-	-
DGF-SQ[2]	N/A	16	5		89.8	79.1	1.71	-	-
GRIDFUN-S	2	16	4	✓	87.1	80.2	**0.09**	5.8	53.14
GRIDFUN-M	3	8	4	✓	88.9	82.6	0.68	8.3	58.99
GRIDFUN-L	3	12	4	✓	89.8	84.3	1.36	9.1	**62.48**
AUTO-DEEPLAB*	5	8	4	✓	83.5	70.0	1.49	12.2	42.47
AMOEBANET**	5/3	12	4	✓	88.1	81.2	1.49	11.2	55.04
AUTODISPNET***	3	18	4	✓	83.2	79.9	6.36	19.3	31.20



$K = 2$



$K = 3$

Fig. 8. The trade-off between model size and accuracy. Results of the NAS are compared to fully re-trained networks. Note that the networks are optimized to reduce the pixel-wise classification loss during training, and during the search process, the networks are selected according to the fitness function $f(x)$ described in Sec. III-E.

evaluation demonstrates that the proposed, automatically configured networks outperform handcrafted fusion networks from literature. Also, the grid fusion backbone populated with cells found by comparable NAS projects from literature show less performance compared to the proposed cell structures.

REFERENCES

[1] C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation,” *CoRR*, vol. abs/1901.02985, 2019. [Online]. Available: http://arxiv.org/abs/1901.02985

[2] T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox, “Autodispnet: Improving disparity estimation with automl,” *CoRR*, vol. abs/1905.07443, 2019. [Online]. Available: http://arxiv.org/abs/1905.07443

[3] A. Elfes, “Occupancy grids: A stochastic spatial representation for active robot perception,” *CoRR*, vol. abs/1304.1098, 2013. [Online]. Available: http://arxiv.org/abs/1304.1098

[4] U. Scheunert, N. Mattern, P. Lindner, and G. Wanielik, “Generalized grid framework for multi sensor data fusion,” in *2008 11th International Conference on Information Fusion*, June 2008, pp. 1–7.

[5] S. Wirges, T. Fischer, J. B. Frias, and C. Stiller, “Object detection and classification in occupancy grid maps using deep convolutional networks,” *CoRR*, vol. abs/1805.08689, 2018. [Online]. Available: http://arxiv.org/abs/1805.08689

[6] M. M. Itkina and M. J. Kochenderfer, “Convolutional neural network information fusion based on dempster-shafer theory for urban scene understanding.”

[7] G. Balazs and W. Stechele, “Deep grid fusion of Feature-Level sensor data with convolutional neural networks,” in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE) (IEEE ICCVE 2019)*, Graz, Austria, Nov. 2019.

[8] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[9] V. John, S. Mita, and H. Tehrani, “Sensor fusion of intensity and depth cues using the chinet for semantic segmentation of road scenes,” 06 2018.

[10] S. Kim, I. Kim, S. Lim, W. Baek, C. Kim, H. Cho, B. Yoon, and T. Kim, “Scalable neural architecture search for 3d medical image segmentation,” *CoRR*, vol. abs/1906.05956, 2019. [Online]. Available: http://arxiv.org/abs/1906.05956

[11] Y. Weng, T. Zhou, Y. Li, and X. Qiu, “Nas-unet: Neural architecture search for medical image segmentation,” *IEEE Access*, vol. 7, pp. 44 247–44 257, 2019.

[12] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” *CoRR*, vol. abs/1709.01507, 2017. [Online]. Available: http://arxiv.org/abs/1709.01507

[13] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” *CoRR*, vol. abs/1707.07012, 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[14] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 459–468. [Online]. Available: http://proceedings.mlr.press/v70/bello17a.html

[15] Z. Zhong, J. Yan, and C. Liu, "Practical network blocks design with q-learning," *CoRR*, vol. abs/1708.05552, 2017. [Online]. Available: http://arxiv.org/abs/1708.05552

[16] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, Eds. Amsterdam: Elsevier, 2018. [Online]. Available: http://nn.cs.utexas.edu/?miikkulainen:chapter18

[17] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *CoRR*, vol. abs/1802.01548, 2018. [Online]. Available: http://arxiv.org/abs/1802.01548

[18] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *CoRR*, vol. abs/1703.01041, 2017. [Online]. Available: http://arxiv.org/abs/1703.01041

[19] T. Elsken, J. Hendrik Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *arXiv e-prints*, p. arXiv:1808.05377, Aug 2018.

[20] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018. [Online]. Available: http://arxiv.org/abs/1807.11626

[21] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *CoRR*, vol. abs/1905.02244, 2019. [Online]. Available: http://arxiv.org/abs/1905.02244

[22] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[23] S. Sun, J. Pang, J. Shi, S. Yi, and W. Ouyang, "Fishnet: A versatile backbone for image, region, and pixel level prediction," in *Advances in Neural Information Processing Systems*, 2018, pp. 760–770.

[24] M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[25] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*.

[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/