Efficient Evolutionary Architecture Search for CNN Optimization on GTSRB

Fabio Marco Johner
Competence Center Electronics (CCE)
Lucerne University of Applied Sciences and Arts (HSLU)
Lucerne, Switzerland
fabio.johner@hslu.ch

Juergen Wassner
Intelligent Sensors and Networks Laboratory (ISN)
Lucerne University of Applied Sciences and Arts (HSLU)
Lucerne, Switzerland
juergen.wassner@hslu.ch

***Abstract*—Neural network inference on embedded devices has to meet accuracy and latency requirements under tight resource constraints. The design of suitable network architectures is a challenging and time-consuming task. Therefore, automatic discovery and optimization of neural networks is considered important for continuing the trend of moving classification tasks from cloud to edge computing.**

This paper presents an evolutionary method to optimize a convolutional neural network (CNN) architecture for classification tasks. The method runs efficiently on a single GPU-workstation and provides simple means to direct the tradeoff between complexity and accuracy of the evolved network. Using this method, we achieved a 11x reduction in the number of multiply-accumulate (MAC) operations of the winning network for the German Traffic Sign Recognition Benchmark (GTSRB) without accuracy reduction. An ensemble of four of our evolved networks competes the winning ensemble with a 0.1% lower accuracy but 70x reduction in MACs and 14x reduction in parameters.

***Index Terms*—ML, CNN, Optimization, EAS**

I. Introduction

In the last few years neural networks have been successfully used for image classification tasks. The architectures of those networks have often been designed by human experts. Nowadays, automated methods to discover such architectures, known as architecture search algorithms, have been developed to reduce the human work required to find a suitable architecture for a given problem e.g. [1], [2], [3], [4], [5], [6].

There are different techniques used in neural architecture search like numerical optimization [2], recursive neural networks (RNN) which propose architectures [4], or search strategies that are based on evolutionary algorithms (EA), often called evolutionary architecture search (EAS). Using such evolutionary techniques, networks that have higher performance than those designed by humans were recently produced in e.g. [7], [8], [9], [10], [11]. However, these approaches require large amounts of computational power (a computer farm with 450 GPUs was used in [8]), which might not always be easily accessible. In this paper we therefore aim at discovering neural network architectures using only modest computing resources.

The current trend to move neural network inference from the cloud to edge devices is motivated by latency, energy and privacy considerations [12]. Since computational power, energy and memory resources are often limited in edge devices, there is an increasing demand for neural networks that have low computational complexity and reduced amounts of parameters. Such reduced networks often exhibit lower classification accuracy which in general results in a performance vs. resource tradeoff [13].

Our main goal is to automatically optimize a neural network architecture using relatively low computing power during the search process. The use of EA is motivated by the promising results obtained in [7] and [8] which show that this method can produce good results. The network is optimized for a specific dataset, which in this work is the German Traffic Sign Recognition Benchmark (GTSRB) [14]. We start from an initial network reference architecture that should be optimized.

The EA is designed to be used with low computational requirements, in our case a single desktop computer with one GPU (see section *Hardware*). This allows the widespread usage of this method as the hardware costs are low.

Our main contributions are:

- We show that the proposed generational EA can be used to optimize neural network architectures for the GTSRB dataset regarding the computational cost and memory requirements for inference. The resulting architectures are comparable to the current state-of-the-art [2].
- We show that it is possible to use EAS with small computational footprint consisting of only one GPU while still achieving good results over days of runtime. This proves the technique feasible if large scale computational resources are not affordable/available and still allows for future upgrades and speedups if desired.
- We further show that it is possible to build ensembles from the network created during the EAS, which outperform the initial reference network while exhibiting lower MACs and parameter count.

The rest of the paper is organized as follows: In section II other work related to this one are mentioned and similarities and differences highlighted. In section III we present our approach and the overall concept with the details described in section IV. In section V the experimental setup is described and results are presented and discussed in section VI. We conclude the paper in section VII.

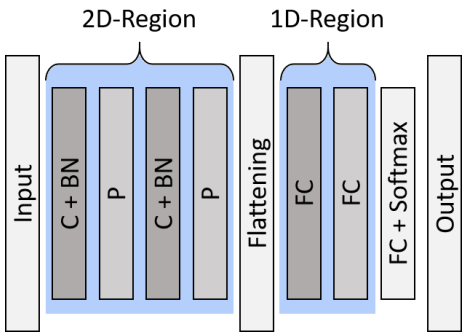

Fig. 1. Overall CNN architecture with optimization regions

II. Relation to other work

As in [7] we employ simple evolutionary techniques and our method doesn't require any human participation once the process is started. Our method creates fully-trained models that are specialized for specific tasks as suggested in [1].

A similar work is [10] where a Monte Carlo algorithm is used to propose network architectures.

Our EA is different from others currently used in EAS, where often a repeated pairwise competition of random individuals is used [7], [8], [9] or the search space doesn't encode the fully-connected layers of the network [11].

Furthermore we don't use parameter sharing like in [6] and create the whole architecture of the network and not cells for cell-based neural network like in [5].

Our round-based EA with increasing selective pressure and stochastic universal sampling can be compared to the aging evolution in [8] as it also avoids to focus too early on good models and thus better explores the search space.

III. Overall Concept and Approach

Our method starts from a reference architecture, yields fully-trained models and allows to easily steer the optimization process towards different goals. Furthermore, the output of the algorithm can be manually post-processed or used as input for other optimization algorithms although this is not a requirement.

A. Search Space

The search space used in this work is based on layers and the common structure of a CNN as illustrated in Fig. 1. The architecture of a neural network starts with the input of an image. This is followed by a number of layers that work on a 2D basis. The number of layers in this "2D-Region" is neither predetermined nor limited. Possible layer types are:

- 2D Convolution (including batch normalization)
- Pooling
- Dropout

Subsequently, the features are transformed from 2D to 1D with the help of a Flattening layer. In the following "1D region" the number of layers is again neither predetermined nor limited. Possible layer types are:

TABLE I
Experimental Search Space used in this work

Layer type	Layer Parameters	Parameter values
2D Convolution	# of kernels	$\in \{1, 2, ..., 49, 50\}$
	stride	$\in \{1, 2, 4, 6, 8, 10\}$
	kernel size	Square, $\in \{3, 5, 7, 9, 11\}$
	padding	valid or same
	activation	linear or ReLU
	dropout usage	true or false
	dropout rate	$\in \{0.25, 0.3, ..., 0.7, 0.75\}$
	batch-norm usage	true or false
Dense/ Fully-connected	# of neurons	$\in \{20, 30, ..., 490, 500\}$
	activation	linear or ReLU
	dropout usage	true or false
	dropout rate	$\in \{0.25, 0.3, ..., 0.7, 0.75\}$
Pooling	stride	$\in \{1, 2, 4, 6, 8, 10\}$
	kernel size	Square, $\in \{2, 3, ..., 10, 11\}$
	padding	valid or same
Dropout	dropout rate	$\in \{0.25, 0.5, 0.75\}$

- Dense/Fully-connected
- Dropout

The optimization parameters for each layer type, together with the supported value range, can be seen in Tab. I.

The final layer of the network is fixed to be a dense layer with the number of neurons corresponding to the number of outputs required by the data set. The activation used in this last layer is the Softmax function to convert the outputs to probability scores for each class label.

The search space defined above constrains the architectural structure of the neural network without limiting the number of layers. This ensures that the same optimization algorithm can be applied to data sets of different sizes. We bound the complexity of the target network by other means, i.e. the maximum number of weight parameters W_{max} and the maximum number of MAC operations M_{max}. More specifically, in this work W_{max} and M_{max} are given by the corresponding values of the initial network in order to ensure lower or equal complexity for the target network. Note however, that this choice is arbitrary and can be adapted to throughput and/or resource constraints of the embedded system at hand.

Our search space defined above is similar to the search space used in [1] but with a stronger overall design pattern. It differs from other work like [8] where the architectures for the cells of a cell-based network are generated instead of the whole network architecture.

B. Evolutionary Algorithm

The EA used in this work is summarized in Algorithm 1. It acts on a population of models in which each individual model represents a neural network architecture. An individual is always created by selecting an existing one, making a copy of its architecture and modifying this copy with a random mutation. In order to not waste computational resources, it is ensured throughout the entire runtime of the algorithm that the same architecture is not created, and thus trained, twice. This is motivated by the assumption that the data set achieves comparable accuracy with every training run, and thus the

Algorithm 1 Evolutionary Algorithm

Require: *refModel, I*, n
  generations $\leftarrow$ *I* ▷ Number of EA cycles
  population $\leftarrow$ *refModel* ▷ Start with reference network
  while $|$*population*$| < n$ **do** ▷ Init Population
    tmpModel $\leftarrow$ COPYANDMUTATE(*refModel*)
    population.add(*tmpModel*)
  end while
  $i = 1$
  for $i <$ *generations* **do** ▷ Main EA loop
    ASSIGNRANKBYMAC(*population*)
    sp $\leftarrow$ GETSELECTIVEPRESSURE(i)
    ASSIGNFITNESSBYRANK(*population*, *sp*)
    parents $\leftarrow$ STOCHUNIVERSALSAMPLING(*population*)
    children $\leftarrow$ COPYANDMUTATE(*parents*)
    TRAINANDEVAL(*children*)
    minAcc $\leftarrow$ CALCULATEMINACCURACY(i)
    tmpSet $\leftarrow$ FILTERBYACC(*children*, *minAcc*)
    tmpSet.add(FILTERBYACC(*population*, *minAcc*))
    population $\leftarrow$ GETBEST(*tmpSet*, n)
    $i = i + 1$
  end for

same architecture is evaluated similarly at different stages of the algorithm.

In order to initialize the EA a population with n trained individuals is needed. For this the reference architecture is taken as the first individual of this initial population. All other $n-1$ individuals are created by making a copy of the reference network and applying a single mutation to it. This can be thought of a close search around the starting point in the search space as no individual is further away from the starting point than one mutation. This seeding of the initial population with a known good model was also used in [9]. The initial population is then trained and evaluated on the reference data set. This results in a test set accuracy for each individual.

The trained and evaluated initial population is the input to the main loop of the EA. First all individuals in the current population are assigned a fitness level. For this the individuals are ranked by their number of MACs where a lower number of MACs results in a better rank. This ranking scheme directs the search for architectures with lower computational demand and is a form of eager search. The fitness of each individual is then determined through linear ranking with selective pressure [15]. The selective pressure (SP) is linearly increased over the predetermined number of iterations of the EA in the range [1.0,2.0]. The lower SP at the beginning of the EA allows a wider exploration of the search space. Over run time the SP gets higher and results in a better exploitation of the individuals in the population. Next a number of individuals are selected as parents by means of stochastic universal sampling [16] to ensure that parents with high as well as low fitness values are selected. The parents are then copied and each copy is randomly mutated to create a set of children with new, untrained architectures. These children are then trained and evaluated. The set of children is then combined with the current population. Each individual is inserted into a set if it achieves a minimal test set accuracy. All individuals in the set are then ranked as described before and the best n selected to form the new population for the next iteration of the EA. Once the predetermined number of generations have been evaluated the EA is stopped.

At this point the EA created, trained and evaluated many neural network architectures which can now be post-processed to determine which one is considered to be the most optimized network or which ones should be taken to form an ensemble.

IV. Method Details

This section complements section III with the details necessary to reproduce our experiments.

A. Mutations

All mutations, selections and values are chosen randomly using a uniform distribution respecting the previously discussed search space. In general a mutation is chosen (e.g. change the kernel size of a 2D Convolution layer) which is then applied to a randomly selected layer (e.g. a 2D Convolution layer). If there is more then one option for a parameter value, the new value is randomly chosen from the set of possible alternatives (see Table I). The exceptions to this parameter mutations are the insert-layer mutations, which add a layer of a selected type at a random location to the network, and the remove-layer mutations, which remove a randomly selected layer from the network.

A mutation can fail during runtime of the EA in three cases. Either the resulting architecture has already been created, the mutation is not applicable (e.g. removing a dropout layer when no dropout layer exists in the network) or the resulting architecture violates the search space restrictions (i.e. W_{max} or M_{max}). If this happens during runtime another mutation is tried.

B. Ranking function

The ranking function is a key element of the EA as it quantifies what is considered a "good" architecture and thus allows to compare two architectures and decide which is the "better" one. The selection process is based on the fitness of each individual, which encodes its rank. Thus, the ranking function controls the search goal and navigates the EA through the big and fairly unrestricted search space.

In this work the initial ranking function resembles an eager search for networks with lower number of MACs. This means that networks that have lower computational cost are considered to be better. To still guarantee a certain performance a minimal test set accuracy restriction is applied to ensure that only architectures with reasonable performance can become a parent in a later generation.

In a second run of the EA the ranking function is replaced with a scalar field that is based on the number of MACs and the achieved test set accuracy of an architecture. The scalar field is a 2D Gaussian function centred at 100% accuracy and $5 \cdot 10^6$

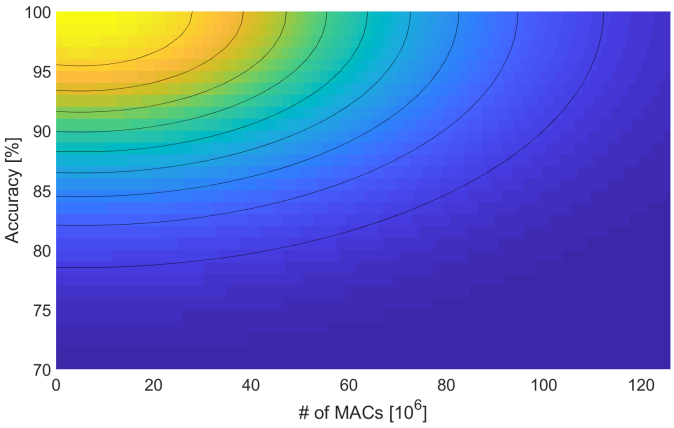

Fig. 2. Scalar field as ranking function (2D Gaussian)

MACs with sigmas of 10% and $5 \cdot 10^7$ MACs respectively, see Fig. 2. It is important to note that this specific function is designed for the GTSRB dataset used in this work (see section *Dataset GTSRB*) but can easily be adapted to fit other datasets.

C. Weight inheritance

We don't use weight inheritance, i.e. no child network is created from an already trained parent network. The rationale behind this is that a good architecture for a given problem should deliver comparable results each time it is trained from scratch.

V. Experimental Setup

A. Dataset GTSRB

The GTSRB dataset [14] consists of a training set with 39'209 real world images depicting 43 different classes of traffic signs and a test set of 12'630 images. We use the whole training set to train a network and evaluate the performance of it with the test set. As the images in the training set vary in size from 15x15 to 250x250 pixels, all images were resized to 48x48 pixels as in [17]. During training we use image augmentation where the images are rotated by $\pm 5°$, shifted by $\pm 10\%$ and zoomed by $\pm 10\%$. This is the same augmentation used during training in [17].

B. Reference Architecture

The reference architecture acts as a starting point for the optimization process. It is represented by the first individual in the initial population and altered with the help of mutations to create the other individuals in the initial population. The winner of the GTSRB challenge [14] was a multi-column deep neural network called MCDNN developed by IDSIA [17]. This MCDNN comprises 25 deep neural networks (DNN) which all have the same architecture. For the classification of an image each DNN calculates their respective output activations which are then averaged together to get the final classification. The achieved accuracy and computational complexity can be seen in the top part of Table II. In our work we use the architecture of the single DNN as the reference architecture to start the EA with. It consists of three convolutional layers with subsequent max-pooling and two fully-connected layers. The difference in the test set accuracy of the DNN from [17] and our work is due to the different batch size of 32 we use in our work. When we retrained the DNN architecture with batch size 1 we could reproduce the test set accuracy (98.42%).

C. Training Details and Hyperparameters

We train each network with the Adam optimizer [18] with a batch size of 32 and an initial leaning rate of 0.0001 for a maximum of 30 epochs. To further improve training performance we use automatic learning rate reduction with a minimum delta of 1% over 5 epochs after which a reduction by a factor of 0.1 is performed. To further reduce the computational load we use early stopping with a minimum delta of 0.5% over 10 epochs.

D. Hardware

A single computer equipped with 32GB of RAM, an i7-8700 CPU and a NVIDIA GeForce RTX 2080 Ti was used to run the experiments. This resulted in a total runtime of the algorithm of around 9 days for the 1350 architectures that were generated and trained. The wast majority of this time was taken by the training of these architectures. Our low-cost setup and low number of trained architectures compares to other recent EAS used on tasks like CIFAR-10 as follows:

- In [8] each experiment ran on 450 Nvidia K40 GPUs and evaluated 20k models in approximately 7 days.
- In [9] they used 270 workers, each equipped with a Google TPU V.2 chip, to train and evaluate 15k models.
- In [4] they used a recurrent neural network (RNN) to generate architectures and utilized 800 Nvidia K40 GPUs for 28 days [3].
- In [3] they used the RNN from [4] in a different setting and used 500 Nvidia P100 GPUs across 4 days.
- A sequential architecture design approach presented in [1] took 8-10 days with a setup of 10 Nvidia GPUs.
- Finally a most recently introduced algorithm [5] to design only the cells for a cell-based network took up to 12 days on a single Nvidia Titan Xp GPU.

E. Search Configuration

The EA was run for 50 iterations with a population size of $n = 50$. In each iteration 13 individuals were selected as parents and from each parent two children were created. This results in a total of 1350 architectures (50 + 50*13*2). $M_{max} = 125'978'700$ and $W_{max} = 1'543'443$ as these are the parameters of the reference architecture from [17] (see Tab. II). The minimal test set accuracy is 92% at the start of the algorithm, then increases raised by 1% after every 10 iterations of the EA.

VI. Results

We used the previously described EA to optimize the architecture of the DNN from [17] on the GTSRB dataset [14] twice. The 1st run uses an eager search for networks with a lower number of MACs and the 2nd run a 2D Gaussian

TABLE II
RESULTS OF EAS AND EVALUATION

Network	Top-1 Acc [%] EAS[a]	Retraining[b]	10-fold cross-val[c]	# MACs [10^6]	# Parameters [10^6]
IDSIA DNN [17]	n.a.	98.47 ± 0.18	n.a.	126.0	1.54
IDSIA DNN[d]	95.92	96.18 ± 0.35	99.89 ± 0.05		
IDSIA MCDNN [17][e]	n.a.	99.46	n.a.	3'149.5	38.59
1st run - eager search (Ranking function: lower # of MACs)					
ID-521	98.73	98.33 ± 0.08	99.94 ± 0.04	47.3	0.80
ID-566	98.64	98.32 ± 0.21	99.92 ± 0.04	15.3	1.12
ID-622	98.29	97.92 ± 0.09	99.87 ± 0.07	19.7	0.75
ID-471	98.07	98.02 ± 0.35	99.91 ± 0.05	15.1	0.93
ID-831	98.02	97.27 ± 0.33	99.90 ± 0.08	5.8	0.66
ID-722	97.95	97.46 ± 0.39	99.87 ± 0.05	5.9	0.70
ID-909	97.89	97.68 ± 0.36	99.90 ± 0.05	5.6	0.18
ID-1073	97.85	97.44 ± 0.17	99.75 ± 0.08	4.8	0.54
Ensemble 1 (831 + 722 + 909)	n.a.	98.86	n.a.	17.3	1.54
Ensemble 2 (566 + 471)	n.a.	98.95	n.a.	30.4	2.04
Ensemble 3 (521 + 566 + 622)	n.a.	99.18	n.a.	82.4	2.66
2nd run - scalar field (Ranking function: 2D Gaussian function, see Figure 2)					
ID-1072	98.60	98.09 ± 0.15	99.85 ± 0.09	11.5	1.11
ID-1162	98.57	98.03 ± 0.12	99.82 ± 0.06	10.9	0.17
ID-916	98.54	97.76 ± 0.13	99.88 ± 0.06	11.3	0.90
ID-1157	98.53	98.12 ± 0.06	99.80 ± 0.12	11.3	0.55
ID-837	98.46	97.85 ± 0.16	99.87 ± 0.06	11.4	0.67
ID-786	98.45	98.18 ± 0.07	99.77 ± 0.06	11.3	0.60
ID-1021	98.16	97.85 ± 0.03	99.81 ± 0.09	8.8	0.54
ID-939	98.15	97.47 ± 0.24	99.76 ± 0.08	9.0	0.76
ID-1343	98.09	97.79 ± 0.12	99.87 ± 0.05	7.4	0.72
Ensemble 1 (1072 + 1162)	n.a.	99.15	n.a.	22.4	1.28
Ensemble 2 (1072 + 1162 + 916)	n.a.	99.28	n.a.	33.8	2.19
Ensemble 3 (1072 + 1162 + 916 + 1157)	n.a.	99.35	n.a.	45.0	2.74

[a]Reached Top-1 accuracy during evolutionary architecture search
[b]The architecture was retrained three times
[c]The architecture was cross-validated three times and the best result took as the metric
[d]Architecture retrained in EAS environment during 1st run
[e]Ensemble of 25 DNN, each trained with differently preprocessed data

function to take into account both, the number of MACs and the accuracy of a network. The results of both runs were examined and a few individuals selected as winning networks of the run. Ensembles were formed based on those trained individuals, with simple averaging of the individual probabilities of the neural networks, as in MCDNN from [17].

All winning network architectures were retrained three times to get a more consistent estimate of their performance. Further 10-fold cross-validation has been performed with each architecture to check that no indirect overfitting occurred during the EA. The retraining resulted in slightly lower accuracies for all winning networks as shown in Table II. This is because their selection is biased towards networks that happened to get optimally trained during EAS. Retraining of lower-performing networks did not show this effect.

The results from the 1st run show that our algorithm can find smaller networks without significant accuracy reduction. For instance, the network ID-909 has a reduction of the retrained accuracy of around 0.8% compared to the reference network, but reduced the number of MACs by a factor of 22.4 and the parameter count by a factor of 8.4. Furthermore, small ensembles can be formed which surpass the accuracy of the reference network while exhibiting a lower MAC count, but have the drawback of an often higher parameter count. Ensemble 1 for example increased the accuracy by around

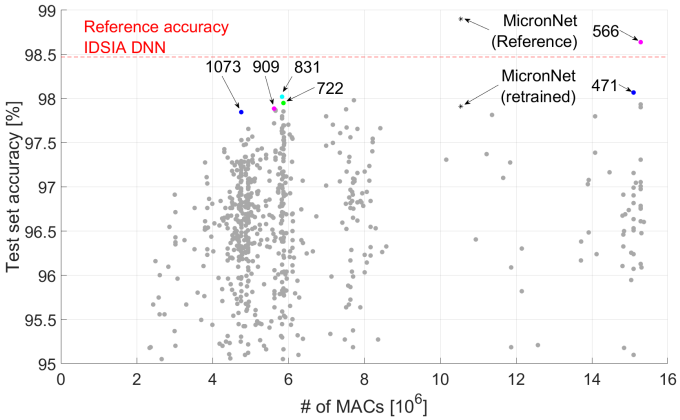

Fig. 3. Point cloud of CNN created during 1st run with a test set accuracy of $\geq 95\%$ and $M \leq 16 \cdot 10^6$

0.4% with a 7x reduction in MACs and approximately equal parameter count. A point cloud depicting the best networks created during this run can be seen in Figure 3.

The results from the 2nd run show even better characteristics which confirms that the optimization goal can be controlled through dedicated ranking functions. Network ID-786 has the best accuracy after retraining of this run (98.18%) with an accuracy reduction of around 0.3% compared to the reference network, although it reached the same accuracy during the

TABLE III
ASSESSMENT OF MICRONNET ARCHITECTURE FROM [2] IN EAS ENVIRONMENT

Top-1-Acc [2]	98.9%
Top-1-Acc[a]	$95.69\% \pm 0.95\%$
10-fold-cross-validation[b]	$99.76\% \pm 0.19\%$
# of MACs	10'543'028
# Parameters	514'979

[a]The architecture was retrained three times
[b]The architecture was cross-validated three times and the best result took as the metric

EAS (98.45%). It exhibits a 11.1x reduction in the number of MACs and a 2.5x reduction in the parameter count. The Ensemble 1, built out of two networks, even surpasses the accuracy of the reference network by around 0.7% with a reduction of 5.6 in MACs and only 83% of the original parameter count. In fact, Ensemble 3 built out of four networks, rivals the much bigger MCDNN from [17], with a 0.1% lower accuracy but with a 70x reduction of MACs and 14x less parameters.

These results can be compared to a current state-of-the-art architecture called MicronNet [2] which was as well optimized for the GTSRB dataset. [2] doesn't use an evolutionary search strategy, but is based on macro-architecture design principles and numerical micro-architecture optimization strategies. The MicronNet architecture has been retrained and assessed in our EAS training environment to allow for a fair comparision. The characteristics and the results from the retraining process can be seen in Table III.

Despite significant effort the stated test set accuracy from [2] of 98.9% could not be reproduced. Different batch-sizes, learning rates and optimizer have been tested, including the ones mentioned in [2], but the best accuracy achieved was 97.91% in one out of 15 runs with a batch-size of 32, automatic learning rate reduction and the Adam optimizer. It is assumed that the accuracy of 98.9% reported in [2] is not based on the network architecture of MicronNet alone but stronger use of data augmentation during training than in [17] or in our work. When the reported accuracy of 98.9% is compared to the results of the two EAS runs, it outperforms every single network. The closest match to it would be Ensemble 1 from the 1st run which has an equal accuracy but twice the MACs and three times the parameter count. On the other hand, when the best accuracy achieved with MicronNet within our training environment is taken as reference, it is outperformed by network ID-909 from the 1st run.

VII. CONCLUSIONS

This work used a generational EA to optimize a neural network architecture on the GTSRB dataset. We created an ensemble (Ensemble 3 from the 2nd run) which competes the much bigger MCDNN from [17], with a 0.1% lower test set accuracy but a 70-fold reduction of MACs and 14-fold fewer parameters. Furthermore, we created an ensemble (Ensemble 1 from the 1st run) which shows that it is possible to build ensembles from the network created during the EAS, which outperform the initial reference network while exhibiting lower MACs and parameter count. Additionally, many good performing network models were generated which provide the user with a variety of architectures as a starting point for further manual optimization (see Figure 3). We run this EAS on a single GPU-workstation to show that EAS can be used with modest computational resources and still achieve good results within days. The architecture search of our algorithm can be guided through dedicated ranking functions to meet different optimization goals. We only used common layer types and no weight pruning to show that the architecture itself is optimized and to assert that resulting architectures can be implemented with current ML acceleration chips in edge devices.

Future work will include the test of the proposed EA on other data sets like CURE-TSR or CIFAR-10.

REFERENCES

[1] B. Baker, O. Gupta, N. Naik and R. Raskar. Designing Neural Network Architectures using Reinforcement Learning. CoRR, abs/1611.02167, 2016.

[2] A.Wong, M.J. Shafiee and M.St. Jules. Micronnet: A highly compact deep convolutional neural network architecture for real-time embedded traffic sign classification. In *IEEE Access*, vol. 6, pp. 59803-59810, 2018.

[3] 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.

[4] B. Zoph and Q.V. Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[5] S. Xie, H. Zheng, C. Liu and L. Lin. SNAS: Stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[6] H. Pham, M. Guan, B. Zoph, Q. Le and J. Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 4095–4104, 2018.

[7] E. Real, S. Moore, A. Selle et al. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.

[8] E. Real, A. Aggarawal, Y. Huang and Q.V. Le. Regularized evolution for image classifier architecture search. In *International Conference on Learning Representations*, 2018.

[9] D.R. So, C. Liang and Q.V. Le. The evolved transformer. CoRR, abs/1901.11117, 2019.

[10] M. Wistuba. Practical deep learning architecture optimization. In *IEEE International Conference on Data Science and Advanced Analytics*, 2018.

[11] L. Xie and A. Yuille. Genetic CNN. In *IEEE International Conference on Computer Vision*, 2017.

[12] V. Sze, Y. Chen, T. Yang and J. S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. In *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, Dec. 2017.

[13] A. G. Howard et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.

[14] J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[15] T. Back. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, 1994.

[16] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the 2nd Annual Conference on Genetic Algorithms*, Massachusetts Institute of Technology, Cambridge, pp. 14-21, 1985.

[17] D. Ciresan, U. Meier, J. Masci and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. In *Neural Networks*, vol. 32, pp. 333–338, 2012.

[18] D.P. Kingma and J.L. Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2015.