USING AN EVOLUTIONARY APPROACH TO EXPLORE CONVOLUTIONAL NEURAL NETWORKS FOR ACOUSTIC SCENE CLASSIFICATION

Christian Roletscheck, Tobias Watzka, Andreas Seiderer, Dominik Schiller, Elisabeth André

Augsburg University
Human Centered Multimedia
Augsburg, 86159, Germany
rolle.roletscheck@t-online.de,tobias.watzka@gmail.com,{seiderer, schiller, andre}@hcm-lab.de

ABSTRACT

The successful application of modern deep neural networks is heavily reliant on the chosen architecture and the selection of the appropriate hyperparameters. Due to the large number of parameters and the complex inner workings of a neural network, finding a suitable configuration for a respective problem turns out to be a rather complex task for a human. In this paper we, propose an evolutionary approach to automatically generate a suitable neural network architecture and hyperparameters for any given classification problem. A genetic algorithm is used to generate and evaluate a variety of deep convolutional networks. We take the DCASE 2018 Challenge as an opportunity to evaluate our algorithm on the task of acoustic scene classification. The best accuracy achieved by our approach was 74.7% on the development dataset.

Index Terms— Evolutionary algorithm, genetic algorithm, convolutional neural networks, acoustic scene classification

1. INTRODUCTION

Deep neural networks (DNNs) have already proven their capability to achieve outstanding performance in solving various classification tasks. Therefore, it is reasonable to use them for acoustic scene classification as well. This trend can be clearly seen in the DCASE Challenge of 2016 [1] and 2017 [2]. However, designing the network architecture and finding the corresponding hyperparameters (learning rates, batch size etc.) remains a challenging and tedious task, even for experts. Therefore, a lot of attention in recent research has been payed on finding ways to automate this process [3, 4, 5, 6]. Among others the *neuro-evolution* method, which relies on evolutionary algorithms (EAs), has been a prominent choice for the task of automatically generating neural networks (NNs). In this work we are exploring the capabilities of neuro-evolution to discover an optimal DNN topology and its hyperparameters for the task of acoustic scene classification. Furthermore, we are introducing our novel self-adaptive EA which uses a genetic representation to create DNNs: *Deep Self-Adaptive-Genetic-Algorithm* (*DeepSAGA*).

2. RELATED WORK

One of the earliest works using EAs to generate NNs, was conducted by Miller et al. [7]. While their approach was originally limited to only evolve the weights of the NN they also showed that it could be advantageous to use an EA to generate a complete NN architecture. In the year 2002 Stanley and Miikkulainen introduced a method called *NeuroEvolution of Augmenting Topologies* (NEAT) which evolves NN topologies along with the weights [8]. Till today NEAT is the basis for many state-of-the-art algorithms in the field of neuro-evolution.

Kroos and Plumbley proposed 2017 in [9] a modified version of the NEAT algorithm. Their EA "J-NEAT" generates small NNs for sound event detection in real life audio. As participants of the DCASE Challenge 2017 they demonstrated that their generated small NNs are able to compete with other bigger networks, such as the baseline approach. Their main concern, however, was the minimization of the total number of nodes used in the generated NN rather than the maximization of the classification performance.

Real et al. [10] have shown in 2017 the capability of EAs to create CNNs solving image recognition tasks. Their fully generated models are capable of competing with the state-of-the-art models manually created by experts. This boost in classification performance, however, comes at the expenses of computational costs. The discovery of the best model took a wall time of roughly 256 hours of evolutionary search, distributed over 250 worker clients. While this approach, therefore, demonstrates the general feasibility of utilizing EAs to discover new DNN architectures it is also largely incapable for most practical applications due to its extensive demand of computation power.

Martin et al. [11] developed a novel EA that evolves the parameters and the architecture of a NN in order to maximize its classification accuracy, as well as maintaining a valid sequence of layers. Parameters related to their EA where empirically chosen by hand and won't change during a run.

The shown state-of-the-art approaches have demonstrated the general feasibility of using EAs to discover new DNN topologies and hyperparameters. However, given their respective specific characteristics, none of them seems to be an optimal fit in our case. Like the other algorithms mentioned here, DeepSAGA evolves the architecture and hyperparameters of a NN as well, while using the backpropagation algorithm [12] for weight optimization. Its main goal lies on the maximization of the classification performance for a given classification problem with a limited amount of disposable compute power. In addition, its own parameters are included in the evolutionary search process, making the search in advance for optimal start parameters obsolete and giving the algorithm the chance to change its parameters autonomously during a run.

3. DEEPSAGA

For the development of our approach, we followed the guidelines set forth by Eiben et al. [13] for designing executable evolutionary algorithms. The creation of an executable EA instance requires

the specification of its parameters. One of their guidelines suggests using parameter control [13, Chap. 7.3] for finding suitable parameters easier, since the resulting values not only influence the finding of an optimal solution, but also the efficiency. In our case, we use the *Self-Adaptive* variant, as it is one of the possible *Parameter-Control* techniques. In this variant the parameters to be adapted are represented as a component of the genetics and are thus part of the evolutionary search space. Therefore, has the potential to adapt the algorithm to the problem while solving it [13, Chap. 8].

The following subsections describe the implementation of our EA, while still taking the guidelines by Eiben et al. into account. The representation and definition of our individuals, the used fitness function and details to our population will be explained in the subsections 3.1, 3.2 and 3.3 respectively. Other subsections in this section are related to the typical steps, that will be excluded by an EA during its run. In the entire course of this work, the term session refers to the holistic process of an EA (from initialization to termination), while the repetition of the steps, selection of parents, recombination, mutation, evaluation of offspring and selection of individuals to form the next population are called a cycle.

3.1. Representation and definition

To enhance readability, we have used a representation analogous to biological genetics. Within biological terminology, a **genome** contains all **chromosomes** and represents the entire genetic of a living being. A chromosome is a bundle of several **genes** contained in the organism, whereby genes determine the different characteristics of that organism.

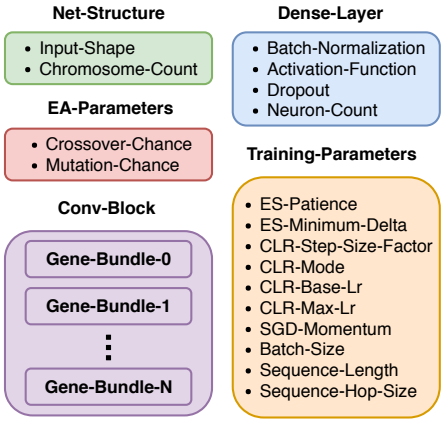

Figure 1: Detailed overview of all chromosomes and genes listed in a genome. ES: early stopping; CLR: cyclic-learning-rate.

In our work, a genome represents the genetic of a NN and describes its characteristics: its architecture and hyperparameters. The **genotype** is expressed by our genome and the **decoding**, in our case, the process of creating and training the network according to the characteristics described by the genotype.

Figure 1 lists our chromosomes contained in each genome. The **Conv-Block** is made up of at least one so-called **Gene-Bundle**. For each Gene-Bundle a convolutional layer followed by a max-pooling layer will be added to a NN architecture. It therefore contains information (genes) about the number of filters, filter-shape and filter-stride. In addition, each Gene-Bundle contains genes indicating whether optional layers for zero-padding, dropout and batch-normalization are included. Finally, a global-average-pooling or flatten layer can be used to connect the Conv-Block with the output or further classification layers.

An **allele** is a concrete expression of a gene. In our chosen representation model, an allele for the **Batch-Size-Gene** could be, for example, the integer number 128. Finally, Figure 2 illustrates the overall design of our genome. An example for a genome with filled in values can be seen in Figure 4, while Figure 5 displays said genome generated CNN architecture.

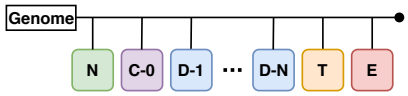

Figure 2: Illustration of a genome. The squarelike objects are representatives for the chromosomes Net-Structure, Conv-Block, Dense-Layer, Training- and EA-Parameters.

3.2. Fitness function

In our case, the focus was mainly on the accuracy of a NN. However, to speed up the evolutionary search process, the number of training epochs of a network was also taken into account. As a result, our $score$ value represents the total fitness of a population member, meaning the higher the $score$ the better the quality of a genotype. The following formula illustrates the utilized fitness function:

$$\mathbf{score} = 0.98 * accuracy + 0.02 * \frac{epoch_{limit} - epoch}{epoch_{limit}} \quad (1)$$

In this context, $epoch_{limit}$ stands for the maximum number of epochs a net is permitted as training time and $epoch$ for the number of epochs with which the net was actually trained. The distribution with 98 % on accuracy and 2 % on the other half, seems to be a solid approach and is solely based on own empirical observations.

3.3. Population

A steady state model [13, Chap. 5.1] is used, to manage the population. To promote diversity and the self-adaptive property, the population size is dynamic. However, since the available resources are limited, the maximum population size is restricted to 90.

3.4. Parent selection mechanism

As described by Bäck and Eiben [14] the parents are determined by a tournament selection procedure. The tournament depends on the population of the current cycle. This also applies to the number of population members who are allowed to participate in the tournament. To calculate said number Formula (2) was used, which takes into account the maximum permitted population size.

$$\mathbf{participants} = popsize_{limit} - popsize_{current} \quad (2)$$

The tournament size is determined by Formula (3), where $toursize_{limit}$ always corresponds to one tenth of the $popsize_{limit}$.

$$\mathbf{toursize} = toursize_{limit} * \frac{popsize_{current}}{popsize_{limit}} \quad (3)$$

If the population limit is reached, the number of participants is limited to **two** until the threshold value is undershot again.

3.5. Variation operators

Mutation

Genes of the category symbolic are mutated by replacing the original allele with a randomly selected. However, the **current** allele has a higher chance of being selected again than the other possible alleles. This type of mutation is also called sampling.

An allele of the integer type is mutated by a creep mutation [13, Chap. 4.3.1] or reset, whose chance is set to 5 %. In our case the sigma value always corresponds to 0.025 (2.5 %) times the limit. For example, if the maximum limit is 1000, the corresponding sigma value would be 25.

Nonuniform mutation [13, Chap. 4.4.1] is used to mutate an allele of the float type. This time, the sigma value is equated with the individual's chance of mutation.

Each population member has its own chance of mutation, which is co-evolved according to the method described in [15]. Before all other genes, the **Mutation-Chance-Gene** is mutated using the nonuniform mutation method. The resulting new mutation chance is the probability with which the remaining genes are mutated.

Recombination

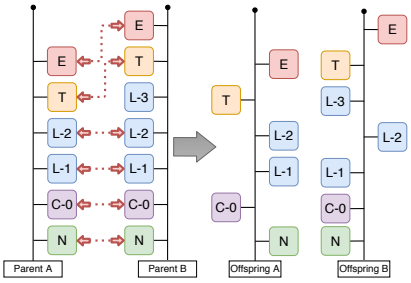

Figure 3: Illustrated procedure of our uniform crossover

The probability with which a recombination (throughout this work referred to as crossover) takes place depends on the crossover chance. Since, in our work, the crossover chance is part of the evolutionary process, it is represented by the **Crossover-Chance-Gene**. Thus, each population member has an individual crossover chance. The recombination process is based on the procedure described in [13, Chap. 8.4.7]. The individual crossover chance p_c of a parent is compared with a random number r ($r \in [0, 1]$). One parent is "ready to mate" if $p_c > r$ applies. This opens up the following possibilities:

1. When both parents are ready to mate, a crossover takes place.
2. If both parents are not ready to mate, they are cloned.
3. If only one parent is willing to mate, a clone of the unwilling parent is created. For the remaining individual a new partner is chosen randomly from the pool of parents, who is also checked for its willingness to mate.

The recombination itself takes place in the style of a uniform crossover [13, Chap. 4.2.2]. For example, offspring A first receives all chromosomes of parent A, then each of the chromosomes of offspring A could be swapped with a chromosome of parent B, taking the crossover chance of parent A into account. The exact procedure is depicted in Figure 3.

3.6. Survivor selection mechanism

Our selection procedure follows an age-based [13, Cap. 5.3.1] replacement strategy. Thus, each newly created individual is assigned a value (remaining lifetime, in short **RLT**) using Formula (4) as described by Bäck [14]. The RLT is reduced by **1** after each cycle, thus determining how long a population member remains alive. Though, the lifetime of the individual with the highest fitness remains unchanged. Where $MinLT\ (\alpha)$ and $MaxLT\ (\omega)$ stand for the permissible minimum and maximum lifetime of an individual. All other variables are linked to the current status of the population. These variables are $fitness\,(i)$, $AvgFit\ (AF)$, $BestFit\ (BF)$ and $WorstFit\ (WF)$. They stand for the fitness of the individual i, the average fitness, the best fitness and the worst fitness of the current population. The prefactor calculation is $\eta = \frac{1}{2} \cdot (\omega - \alpha)$.

$$\mathbf{RLT}\,(i) = \begin{cases} \alpha + \eta \cdot \frac{WF - fitness(i)}{WF - AF} & \text{if } fitness\,(i) \geq AF \\ \frac{1}{2}\,(\alpha + \omega) + \eta \cdot \frac{AF - fitness(i)}{AF - BF} & \text{if } fitness\,(i) < AF \end{cases} \tag{4}$$

The authorized minimum and maximum lifetime of an individual has been set to **1** and **7**. If the fitness value of a newly created individual i is better than the average fitness, it receives a lifetime from **5** to **7**, otherwise a lifetime from **1** to **4**. Within these subareas, the better individuals have a longer lifespan than the individuals with a lower fitness.

3.7. Initialization and termination

The individuals of the first population are generated randomly. Since the available resources are limited, the expressions of the respective genes are bounded. These limits must not be exceeded by variation operators either. The termination criterion is the completion of the **40th** cycle, considering the optimum is, in our case, not known in advance.

4. EXPERIMENTS

4.1. Setup

To evaluate the proposed genetic algorithm we use the TUT Urban Acoustic Scenes 2018 dataset from subtask A provided by the DCASE 2018 Challenge [16]. The dataset consists of 10-seconds audio segments from 10 different acoustic scenes. For every acoustic scene, audio was captured in 6 different cities and multiple locations. To train and measure the performance of the generated models we use the development dataset with the suggested partitioning for training and testing.

To generate the input features for the NNs the stereo audio samples were first converted into mono channels. Thereafter, the librosa library (v0.6.1) [17] was used to extract log mel spectrograms with 100 mel bands that cover a frequency range of up to 22050 Hz. For the Short-Time Fourier Transform (STFT) a Hamming window with a size of 2048 samples (43 ms) and a hop size of 1024 samples (21 ms) was used. The resulting spectrograms were divided into sequences with a certain number of frames defining the sequence length. For the creation of the sequences an overlap of 50 % was used. The sequence length can vary depending on the different models generated by the genetic algorithm.

Due to the stochastic behaviour of the algorithm, two independent sessions of 10 cycles each were initially completed. After-

wards, the best 30 models from each of these sessions were added to the initial population of a final third session. This approach results in a population with a higher initial fitness while keeping a certain degree of diversity. In order to speed up the process as a whole, several computers were connected in a client-server concept manner. The server distributes the genotypes from the current cycle population to all available clients, on which side the decoding takes place. Altogether, 15 clients, each equipped with an NVIDIA GTX 1060 graphic card, were available for the NN training process. Therefore, depending on the current population size, complexity of the genomes and number of available clients a cycle took around two to three hours. Overall the elapsed wall time was roughly 120 hours, 87 hours if excluding the two initially completed sessions.

At the end of the final session, the best NN was used for classification. In addition, an ensemble learning strategy was pursued. From a cycle the **10** best individuals could also be selected to vote together on the class of an audio sample. Individuals in higher ranks have more votes to weight them higher. Finally, the class with the most votes wins. In this paper this type of classification is referred to as **population vote**.

4.2. Results

Table 1 illustrates the final results. At the end our best CNN ("DeepSAGA CNN") reached an average accuracy of 72.8 % on the test subset of the development dataset. For the population vote ("Pop. vote") strategy, on the other hand, an average accuracy of 74.7 % was reached.

Scene label	DCASE2018 Baseline	DeepSAGA CNN	Pop. Vote
Airport	72.9 %	84.9 %	85.7 %
Bus	62.9 %	63.2 %	67.4 %
Metro	51.2 %	71.3 %	71.6 %
Metro station	55.4 %	75.3 %	81.9 %
Park	79.1 %	81.0 %	82.2 %
Public square	40.4 %	53.2 %	56.0 %
Shopping mall	49.6 %	75.3 %	73.8 %
Street, pedest.	50.0 %	67.2 %	69.6 %
Street, traffic	80.5 %	85.0 %	86.2 %
Tram	55.1 %	72.0 %	72.4 %
Average	**59.7 %**	**72.8 %**	**74.7 %**

Table 1: The class-wise accuracy for task 1 subtask A evaluated on the test subset of the development dataset.

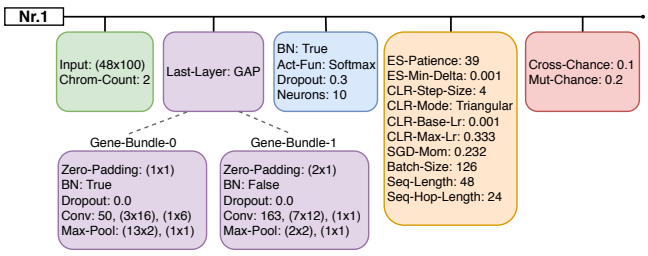

Figure 4: Best genome of the session. GAP: Global-Average-Pooling; BN: Batch-Normalization. The numbers in the brackets are the filter size and the filter stride for the convolutional and max-pooling layers and the first number for the convolutional layer stands for the number of filters.

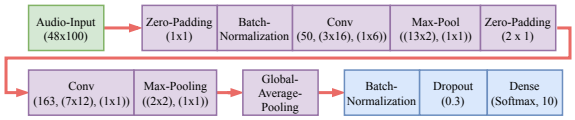

Figure 5: Architecture of "DeepSAGA CNN".

The genome of the "DeepSAGA CNN" can be seen in Figure 4. Figure 5 displays the generated architecture (taking its genome into account) of said CNN. The used hyperparameters can be found in its **Training-Parameters-Chromosome**, which is visible in Figure 4

5. DISCUSSION AND CONCLUSION

In this paper, we described how we developed a genetic algorithm called DeepSAGA to automatically generate CNNs from scratch. Once a session is started, it can be left unattended and offers a selection of NNs after it terminated. We used the DCASE 2018 Challenge as an opportunity to evaluate our algorithm for its competitive ability. With an accuracy of 74.7 % on the test subset the algorithm showed promising results with this specific dataset.

Our investigations throughout an entire session showed that the inclusion of hyperparameters in the search process was an important decision. With regard to their hyperparameters, it often happened that a clone differed (only in its Training-Parameter-Chromosome) from its original only in a few places, but still led to a clear difference in their accuracy. These observations suggest that architectures probably only work well with certain hyperparameter constellations and hyperparameters only with certain architectures.

Throughout the sessions, the approach of population vote resulted in a higher accuracy than that of the best model of the corresponding cycle. A possible reason for that, could be the nature of the population vote itself. Selecting the 10 best CNNs from a cycle, results in predicting with different suitable architectures and hyperparameters simultaneously, while weighting the votes of models with a better fitness higher. Thus, leading to a better overall accuracy as they counterbalance their weaknesses. However, there were fluctuations in the accuracy difference.

In each of our generated CNNs, the "public square" class always proved to have the lowest detection rate. This phenomenon is also reflected in the baseline. A closer look revealed that this class is often mistakenly recognized as a "shopping mall" or "street traffic". In all of these classes background talking is existing and except for the shopping mall traffic noise is involved. Except for adding additional data in future work it could be researched how the evolutionary algorithm could be improved to especially handle the problem with such classes.

Currently DeepSAGA is limited in that way that the architecture after the input layer consists of a series of convolutional layers followed by a series of dense layers. This limitation means that an architecture such as a convolutional layer followed by a dense layer followed by a convolutional layer cannot be created. Additionally, architectures with recurrent layers were not included. Both additions could increase the classification accuracy but also introduce a vast of new parameters that have to be tested by the algorithm if no restrictions are made.

The approach of DeepSAGA is generic and not limitied to audio. Nevertheless, in the future further investigations are required to evaluate its performance with other types of data and data sets so that it can be further optimized to get nearer to the goal to make handcrafted NN architectures obsolete.

6. REFERENCES

[1] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection." IEEE, 2016, pp. 1128–1132. [Online]. Available: http://ieeexplore.ieee.org/document/7760424/

[2] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, 2017, pp. 85–92.

[3] M. Kim and L. Rigazio, "Deep clustered convolutional kernels," vol. abs/1503.01824, 2015. [Online]. Available: http://arxiv.org/abs/1503.01824

[4] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[5] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra, "Convolution by evolution: Differentiable pattern producing networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. ACM, 2016, pp. 109–116. [Online]. Available: http://doi.acm.org/10.1145/2908812.2908890

[6] G. Morse and K. O. Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. ACM, 2016, pp. 477–484. [Online]. Available: http://doi.acm.org/10.1145/2908812.2908916

[7] G. F. Miller, "Designing neural networks using genetic algorithms." 1989.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," vol. 10, no. 2, pp. 99–127, 2002.

[9] C. Kroos and M. Plumbley, "Neuroevolution for sound event detection in real life audio: A pilot study," in *DCASE 2017*, T. Virtanen, A. Mesaros, T. Heittola, A. Diment, E. Vincent, E. Benetos, and B. Elizalde, Eds. Tampere University of Technology, 2017. [Online]. Available: http://epubs.surrey.ac.uk/842496/

[10] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," 2017. [Online]. Available: https://arxiv.org/abs/1703.01041

[11] A. Martn, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "EvoDeep: A new evolutionary approach for automatic deep neural networks parametrisation," vol. 117, pp. 180–191, 2018.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," vol. 323, p. 533, 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[13] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer Berlin Heidelberg, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44874-8

[14] T. Bäck and A. E. Eiben, "An emperical study on GAs without parameters," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2000, pp. 315–324. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45356-3_31

[15] T. Bäck, "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm." in *PPSN*, 1992, pp. 87–96.

[16] A. Mesaros, T. Heittola, and T. Virtanen, "A multi-device dataset for urban acoustic scene classification," 2018. [Online]. Available: http://arxiv.org/abs/1807.09840

[17] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, 2015, pp. 18–25. [Online]. Available: http://www.academia.edu/download/40296500/librosa.pdf