Computing and Informatics, Vol. 42, 2023, 340–363, doi: 10.31577/cai_2023_2_340

EVOLUTION-BY-COEVOLUTION OF NEURAL NETWORKS FOR AUDIO CLASSIFICATION

Włodzimierz Funika, Paweł Koperek, Tomasz Wiewióra

Institute of Computer Science
Faculty of Computer Science Electronics and Telecommunication
AGH University of Krakow, al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: funika@agh.edu.pl, {pkoperek, tomasz.wiewiora95}@gmail.com

Abstract. Neural networks are increasingly used in recognition problems, including static and moving images, sounds, etc. Unfortunately, the selection of optimal neural network architecture for a specific recognition problem is a difficult task, which often has an experimental nature. In this paper we present the use of evolutionary algorithms to obtain optimal architectures of neural networks used for audio sample classification. We extend the Pytorch DNN Evolution tool implementing co-evolutionary algorithms which create groups of neural networks that solve a given problem with a certain accuracy, with the support for problems in which training data consists of audio samples. In this paper we use the co-evolutionary approach to solve a sample sound classification problem. We describe how the sound data was prepared for processing with the use of the Mel Frequency Cepstral Coefficients (MFCC). Next we present the results of experiments conducted with the AudioMnist dataset. The obtained neural network architectures, whose classification accuracy is comparable to the classification accuracy attained by the AlexNet neural network, and their implications are discussed.

Keywords: Neural networks, evolutionary algorithms, sound recognition

Mathematics Subject Classification 2010: 68-T05, 68-T10

1 INTRODUCTION

Speech is the basic medium of interpersonal communication. However, it is more and more often used as a communication channel between a human and a ma-

chine thanks to recent developments in the area of speech recognition. Nowadays, we observe a very rapid development of many tools based on speech recognition technology. There are multiple examples of such systems, e.g. virtual assistants ([1]: Google Assistant, Apple Siri, Amazon Alexa, Microsoft Cortana), car control systems [2], robot control systems [3]. The number of applications and systems which employ a speech-based interface is constantly growing. Speech recognition and audio classification are already used in search engines, car navigation and translators.

Recently, using the neural networks became a very popular approach to creating audio classification systems. One of the main challenges in this context is the time consuming process of designing the neural network architecture. It requires a lot of domain knowledge and a large number of experiments. Incorrect decisions may lead to suboptimal classification performance and render the newly created systems incapable of serving its basic purpose. In order to automate and streamline the model discovery process an evolutionary algorithm may be used. In this paper we demonstrate how the Pytorch DNN Evolution framework [4] can be used to accelerate the process of creating neural network architectures which solve the audio classification problem. The network architectures are obtained in subsequent iterations of the genetic algorithm, which over time solves the given classification problem. To validate our approach we present the results of a series of experiments conducted with the AudioMNIST dataset [5] used as a sample input dataset.

The neural network model is only one of the components required. Audio preprocessing is another crucial element of building a system which communicates with human users successfully. However, the sound signal analysis is also applicable to many other fields, e.g. medicine, bio-acoustics or seismology. In medicine, mainly in otorhinolaryngology, a spectrogram could be used in a voice examination. It separates the sound signal into bands with different frequencies. Such a result is used by a phoniatrist to detect a subtle early changes in the voice. These changes may be the initial stage in the development of vocal chords nodules [6]. Another field in which sound signal analysis is used is bio-acoustics – it studies the impact and role of sound in the lives of animals. In this field, tools are mainly used to extract individual sound characteristics, which can be used to distinguish between species of animals or even their specific individuals. An example of the use of sound recognition techniques in bio-acoustics is a bat echolocation research [7]. Sound analysis is also often used in seismology. There are methods for extracting features from sound samples. Systems for the classification of micro-seismic signals are being developed in order to minimize risks in the mining industry [8]. These systems aim to early detect events which might cause dangerous vibrations in mines. Since audio pre-processing can be used in such a variety of contexts, there are numerous available techniques. This means that audio pre-processing requires careful examination and adjusting to a specific problem. In the system presented in the current paper, the audio signal is transformed to the MFCC coefficients [9]. In our research we have attempted to empirically determine an optimal number of coefficients which need to be used in the context of the analyzed dataset.

The paper is structured as follows. In Section 2 we discuss related research and the background of our work. In Section 3 we describe the modifications to the Pytorch DNN Evolution framework. In Section 4 we present a description of the conducted experiments. In Section 5 the results of the experiments are discussed. Finally, in Section 6 we conclude our research based on the conducted experiments.

2 RELATED WORK

In this section the background for our research is presented.

2.1 Using Neural Networks in Sound Classification

Neural networks are a very flexible method for approximating very complex functions. This makes them very useful in many domains, including speech recognition. An example of the use of neural networks in those areas can be the problem of classifying the sound recordings of numbers in English [10]. The authors presented a method of sound samples classification based on spectrograms. The architecture used in the experiment was based on the architecture of AlexNet [11]. The architecture contained five convolutional layers. Two types of experiments were conducted on the AlexNet network. The first one was a classification of the recordings of digits, so there were ten possible classes. The second one was classification of the recordings according to the gender of the person who has been recorded.

The problem of sound classification can also be solved with architectures, which previously worked well with the problems of image classification [12]. The authors adapted the existing network architecture *VGG19* [13]. The *VGG19* architecture is most often used to classify images. In the case of image classification tasks, it is common only to retrain the fully connected layers. However, in the case of audio classification, the authors decided to retrain the last convolutional block along with the fully connected layers.

The above approach is based on manual adaptation of the network architecture to a new type of problem. However, this does not guarantee the creation of an optimal architecture that will solve the classification problem embedded in another domain. Subsequent changes introduced in such a network architecture and their subsequent verification are time-consuming. Therefore, we would like to propose an approach to automating this process with the use of the coevolutionary algorithm outlined below.

2.2 Convolutional Neural Networks

There are many types of neural networks which are used in the sound recognition problems. Convolutional Neural Network *(CNN)* [14] is one of the most widely used ones. *CNN* typically consists of four types of layers [15]: convolutional layer, pooling layer, fully-connected layer (dense layer), and a softmax layer.

All neurons in the convolution layer take as input a cross-section of the output from the previous layer. Each neuron multiplies the local input data by the weight matrix. The weight matrix or the local filter is replicated over the entire input space to detect a specific type of pattern. All neurons share the same weights to create a feature map of objects. The entire convolutional layer consists of many feature maps of objects that have been generated using differently placed filters. This procedure is used to isolate many types of local patterns that may occur in any location. For speech recognition, the input space may be a two-dimensional plane where the dimensions of the data are frequencies and time [16]. Following the convolution layer typically a pooling layer is used. Such a layer similarly as the convolutional layer takes input from the local region of the previous convolutional layer to generate a single output from that region. The common operator of these layers used in *CNN* is *max-pooling*. It outputs the maximum value in each sub-region. This operation reduces the computational complexity and makes the network resistant to slight changes in the position of local patterns. The next layer after at least one convolutional and one pooling block is a fully connected layer, also called *dense layer*. The main task of this layer is the final classification of the object. This layer identifies the input object and assigns it to the specific class.

In literature one can also find other types of neural network types, e.g. recurrent [17], LSTM [18] or Transformer [19]. They have various applications, however in this paper we focus just on those which are relevant to our research.

2.3 Evolving of Deep Neural Network Architectures

The use of Deep Learning provided state-of-the-art results in many domains like image recognition [11, 13] or machine translation [20, 21]. Unfortunately, it involves to carefully design the neural network architecture, which is often a very complicated task. It relies heavily on the experience and knowledge of the researcher, what makes it difficult for beginners to modify or create new models fitting their particular use-case. A variety of Neural Architecture Search (NAS) algorithms were developed [22, 23] to tackle this issue. Among the employed methods there are examples of the Reinforcement Learning [24, 25], gradient optimization [26] or evolutionary algorithms (EA) [27, 28]. In the current research we focus on the last category, due to the high flexibility of the algorithms from that category.

Evolutionary algorithms share one common weakness: in their basic form they require large amounts of resources in order to evaluate the architectures they create. In order to mitigate this problem different strategies are being employed [22]:

- *Reducing the search space* [29, 27]. In the basic approach, the search space of architectures has the representation of all necessary components of an architecture (e.g. layers, their sizes, connections between layers etc.). This means that the algorithm needs to take many smaller steps in order to arrive at the optimal solution. To reduce the number of such steps, less granular concepts are

introduced as search space building blocks, e.g. *cells* or *blocks* (which consist of multiple neural layers themselves).

- *Reusing neural architecture* [28]. Instead of creating every model from scratch, the search procedure uses the existing artificially designed architectures as a starting point, then it and transforms them to improve the performance.
- *Incomplete training* [30]. The individual architecture evaluation is speeded up by changing the mechanism which ranks the architectures between each other. Instead of conducting a lengthy training of a full dataset, e.g. early stopping can be used to limit the amount of computations required to obtain the result of a comparison. Another variant of this approach is to share some or all weights with an already trained model.

The co-evolutionary approach employed in the current paper [31] uses subsets of the original training dataset to reduce the amount of time required to evaluate an individual. Low level architecture building blocks (neural network layers) are used to define the search space. All individual neural network models are created and trained from scratch. This allows to categorize the method as using *incomplete training* technique while not employing *neural architecture reuse* nor *reducing the search space*.

2.4 Coevolution of Neural Networks and Fitness Predictors

Coevolutionary algorithms are a type of evolutionary algorithms in which the training process involves two or more individuals species. In the coevolution algorithm, the assessment of the quality of a given individual in a population depends on individuals from a different population. Coevolutionary algorithms can be divided into two main types: *competitive* and *cooperative* ones. In the competitive approach, the assessment of an individual is obtained through competing with the individuals of the other population. On the other hand, the cooperative algorithms allow individuals from one population to enhance the fitness of individuals of the second one. This means they promote cooperation of individuals with each other. During the training process, this results in rewarding the individuals for solving problems together and punishing for independence.

The evaluation of the individuals in the context of the evolutionary algorithms can be one of two types: objective or subjective fitness. The first one is aimed at defining a function that is used to evaluate the assessment of a given individual which does not require taking into account other individuals (e.g. an error rate in classification of images). In the second approach the assessment depends also on other individuals, including individuals of a different species and thus can be classified as using subjective fitness. To evaluate neural networks (first species), training set subsets (second species) are being used. Such an approach can be used to avoid using the full, very often large, training dataset to evaluate the neural networks what helps to significantly reduce the time required for such an evaluation.

An example of implementation of this idea is the Pytorch DNN Evolution [31] project which attempts to discover an optimal architecture of a neural network used for the purpose of image classification. It conducts the process of co-evolution of two populations in which it recombines and mutates individuals to obtain the most fit individual. The first population consists of neural networks architectures, which are being trained to classify images. The second one is a population of subsets of the original training examples dataset, which can be used for quick assessment of the neural network model. In other words it is a population of so called *fitness predictors*. Each population has its own definition of *fitness*, which is a measure of *quality* of an individual and is used to select which individuals are going to survive to the next iteration of evolution. It is worth nothing that hyperparameters of the neural network training process are provided as the input to the process and are not modified by the evolution.

It is important to note that the *fitness* definition in the context of evolution of neural networks depends on the problem which is being solved. For a sample task of predicting the next element in a time-series it might be, e.g., the accuracy of predictions made by the neural network. In Pytorch DNN Evolution each population uses its own fitness definition. When trained with the use of dataset S, the neural network's n fitness $f_{NN}(n, S)$ can be defined as the accuracy of image recognition on the given set examples. This can be formalized as Formula (1).

$$f_{NN}(n, S) = 100 \cdot \frac{\sum_{i=1}^{|S|} \text{IsCorrect}(\text{Recognize}(n, x_i), \text{Target}(x_i)}{S}, \tag{1}$$

where:

- S – the training dataset, it can be e.g. the full dataset or a fitness predictor,
- $|S|$ – is the size of the dataset S,
- x_i – i^{th} element of dataset S,
- $\text{Recognize}(n, x_i)$ – is the class recognized by the neural network under evaluation,
- $\text{Target}(x_i)$ – is the expected class of the sample as specified in the dataset,
- $\text{IsCorrect}(x, y)$ – is a function which can be defined as follows:

$$\text{IsCorrect}(x, y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

As the evolution progresses, we expect the fitness to increase, which translates to an improvement to the recognition accuracy. In the second population of fitness predictors, the objective of the evolutionary process is to find a subset of the training dataset which allows to compare the fitness of two neural networks. One way to achieve such a goal is to identify the samples of the training dataset, which render similar results to training over the complete dataset. Such samples might have features which are e.g. very common across the dataset or might be very difficult to

accurately recognize. The fitness of a fitness predictor p (the $f_{FP}(p)$) is therefore also defined with the use of the recognition accuracy, however in this case it is not maximized (Formula (3)).

$$f_{FP}(p) = 100 * |f_{NN}(T, p) - f_{NN}(T, \text{FullDataset})|, \tag{3}$$

where:

- FullDataset – the full training dataset,
- p – fitness predictor under evaluation, subset of FullDataset,
- f_{NN} – the neural network fitness as defined in Formula (1),
- T – the neural network used as a trainer for the fitness predictor population.

The co-evolution algorithm which implements these ideas is expressed more formally in the form of a pseudocode, presented in Listing 39.

The result of the coevolution algorithm is a set of neural network architectures. In the paper [31] the described algorithm is applied to the image recognition problem based on the *MNIST* [32] dataset.

2.5 Sound Representation

Sound can be represented in many ways [33]. Depending on the needs, various representations of the sound samples allow to emphasize a specific aspect of the data that is the most interesting in a given context. The most basic method to represent audio data is the waveform which describes changes in sound amplitude over time. Such a representation is very easy to interpret by humans. Another method used to represent audio data is the spectrogram. It shows the distribution of the amplitude spectrum of the sound signal at a given time. Thus, it informs us about the distribution of the intensity of the sound components depending on the frequency of these components. One of the most popular representations is the Mel Frequency Cepstral Coefficients (*MFCC*) [9]. It is based on the *mel* scale. This scale determines the subjective perception of the sound level by human due to the frequency measurement scale measured in hertz (Hz). The units of this scale are called *mel*s. *MFCC* is often used to prepare sound data as input for neural networks. It was used in the preprocessing described in [34]. Each recording has been split into audio chunks and transformed into the *MFCC* representation.

3 EVOLUTIONARY SYSTEM MODIFICATIONS

The main functionality of the Pytorch DNN Evolution framework [4] is the automatic discovery of neural network architectures for solving supervised learning problems. It has been designed to support only datasets consisting of images, e.g. the *MNIST* [32] and *CIFAR10* [35] collections, which are examples of the image classification problems. However, the architecture discovery method implemented

```
def EvaluateIndividual(dnn, fp):
  phenotype = TranslateGenotypeToDNN(dnn)
  dataset_sample = ExtractSamplesFromTrainingDataset(fp)
  phenotype.train(dataset_dample)
  return phenotype.evaluate_test_dataset()

def EvolutionIteration(parents, trainer):
  children = []
  parents_size = len(parents)
  for i in range(parents_size):
    swap(parents, i, random(parents_size))
  for i in range(0, parents_size, 2):
    crossed_over = CrossingOver(parents[i], parents[i+1])
    mutated = [Mutate(crossed_over[0]), Mutate(crossed_over[1])]
    children.extend(mutated)

  for i in len(children):
    children[i].fitness = EvaluateIndividual(children[i], trainer)

  new_population = TournamentPopulations(parents, children)
    best_individual = SelectBestFitnessIndividual(new_population)
    return new_population, best_individual

def CoEvolution(N_fp, N_dnn, N_epochs):
    population_fp = InitializeRandomFPPopulation(N_fp)
    population_dnn = InitializeRandomDNNPopulation(N_dnn)

    best_fp = RandomInt(N_fp)
    best_dnn = RandomInt(N_dnn)

    for i in range(N_epochs):
      for j in range(N_dnn):
        population_dnn, best_dnn = EvolutionIteration(
          population_dnn, best_fp)
        best_dnn.fitness = EvaluateIndividual(
          best_dnn, FullTrainingDataset)
        population_fp, best_fp = EvolutionIteration(
          population_fp, best_dnn)
```

Listing 1. The pseudocode of the co-evolution algorithm implemented by *pytorch-dnn-evolution* package

in the discussed framework, presented in Figure 1, is not constrained to that class of problems. It can be applied to other domains, which can be expressed as supervised learning problems, i.e. it is possible to create a dataset for which sample network outputs can be assigned. In the current paper we present an attempt to use the the Pytorch DNN Evolution for sound recognition domain. In the sections that follow we demonstrate its effectiveness by applying it to a sample dataset.

The Pytorch DNN Evolution is designed to work in a distributed environment and consists of two major components: the *evolution driver* and *workers*. The first component is responsible for execution of the evolutionary algorithm. It maintains two populations: generates the genotype of the individuals constituting the initial population, crosses-over and mutates them according to set probability parameters, triggers evaluation of individual's fitness when necessary. The responsibility of the worker is to perform evaluation of an individual, what can be translated to the

following steps: translating the genotype to a trainable neural network, preparing the input dataset for training, conducting the training and finally measuring and reporting the fitness value, e.g. by testing the classification accuracy with the use of a separate test dataset. It is worth noting that to perform all of its tasks, the evolution driver is not required to translate the individual's genotype to another form. Since the genotype is represented as an array of numbers, applying crossing-over and mutation is a straightforward operation. Thanks to that, the evolution driver component can be applied to a wide range of problems and does not need to be changed, e.g., to introduce support for other types of neural network layers or a new domain. On the contrary, it is just the worker component which needs to be altered. Such an architecture allows the researcher to focus on the details of a specific domain and allows to simply reuse the core co-evolutionary algorithm without changes.

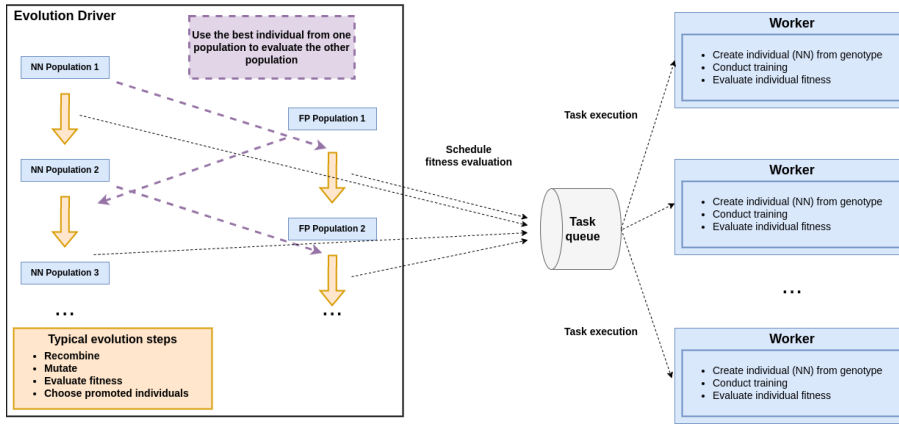


Figure 1. The architecture of the pytorch-dnn-evolution

In our work, to allow the application of the co-evolutionary approach to the domain of sound classification, we have extended the worker component. The worker is one of the crucial parts of the evolutionary system, as it conducts the evaluation of the individual genotypes. The modifications included:

- Interpretation of the individual genotype has been extended with support for other neural network layer types e.g. a convolutional layer. The network architecture is is not limited to creating simple, fully connected layers anymore. Supporting the new layer types includes also dynamically introducing the additional components which transform the format of samples between layers.
- Adjusting the training logic. Introducing the support for different layer types required also changing how the training and evaluation of networks is conducted.
- Introducing support for new types of datasets. This involves extending the preprocessing procedures to ensure that the data samples are presented to the neural

network as Mel Frequency Cepstral Coefficients and the datasets can be used to conduct co-evolution. Operations on audio data such as reading data from a file and retrieving *MFCC* were implemented using the *torchaudio library* [36]. In order to allow evaluation of the discussed approach, we have chosen to integrate the AudioMnist dataset [5], which has been already widely used in that research area [37, 38].

The modified version of the framework is presented in Figure 2.

During the first attempts to train neural networks on a set of audio data, we found a considerable time overhead was introduced by the pre-processing of sound data (processing the relevant part of the dataset into MFCCs). This would have a significant impact on the overall operating time of the genetic algorithm, since each neural network training process requires pre-processing of data before it can commence. Therefore, we have implemented the caching of the MFCCs and labels associated with these data. In this approach, the worker converts the entire audio data set to MFCCs only once before the first training. Coefficients are stored in the file with the relevant labels. During the training of the subsequent neural networks, the worker uses the data saved in this file. This optimization allowed to significantly reduce the experiment time.

4 DATASET PREPARATION

To conduct our experiments we have used the AudioMnist dataset. This collection contains 30 000 recordings of reading numbers from 0 to 9 in English. The recordings were prepared by 60 various speakers. Each recording is additionally enhanced with metadata about the speaker, such as: accent, age, gender. 48 men and 12 women participated in the recordings. The dataset could be used as a model benchmark for various audio data classification tasks. The MNIST or the CIFAR10 datasets perform a similar function for the classification of images.

The analysis of Mel Cepstral Coefficients was used for the data preprocessing for neural networks. The *MFCC* is based on the mel scale that reflects the subjective perception of sound, which is often used in the audio analysis. Generating MFCCs requires choosing an appropriate number of coefficients which are taken into account when analyzing a sound sample. The result of *MFCC* analysis is a three-dimensional representation of the recording. The dimensions of this data type are time, the number of Mel Cepstral Coefficient and its value [9]. Figure 3 presents the results of such an analysis for 10 sound samples of English words from *zero* to *nine*. The horizontal axis represents time, the left vertical axis – the number of coefficients, the color denotes the MFCC value.

As depicted in Figure 3, the values of Mel Cepstral Coefficients oscillate closer and closer to zero as the frequency increases. This means that they are less and less useful for the neural network training. Unfortunately, at the same time, processing them requires using a model with more parameters, what leads to an increase in the training time. To optimize the training time, only a subset including between 12

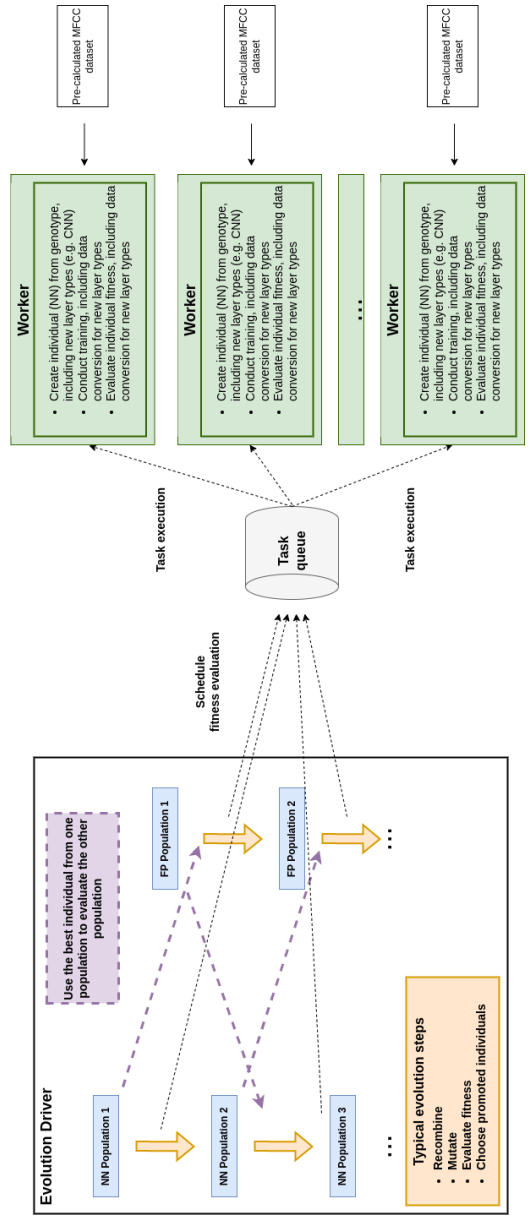


Figure 2. The extended architecture of the pytorch-dnn-evolution which includes changes described in Section 3

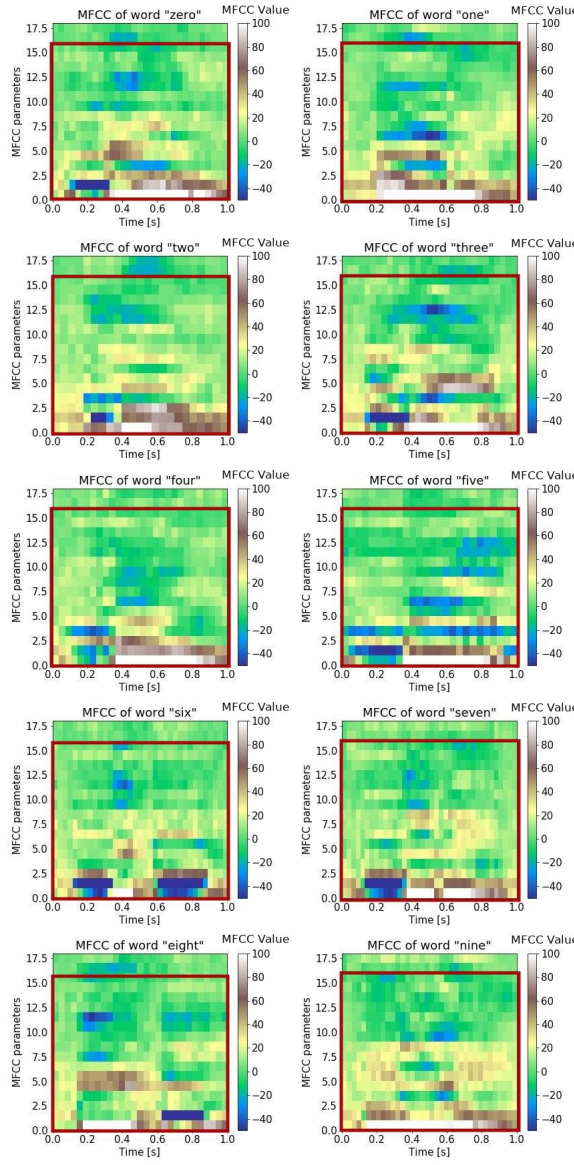


Figure 3. Visualization of the first 18 *MFCC* mel coefficients of the audio sample of words from 'zero' to 'nine'. The coefficients that were taken into account for neural network training were marked with a red rectangle.

to 18 first *MFCC* coefficients is taken into account for the purpose of training. The optimal number of coefficients to be taken into account has been determined empirically by running the neural network training experiments. In those experiments the neural network consisted of three convolutional layers. The dataset was divided into a training set (25 000 samples) and a test set (5 000 samples). The charts in Figures 4 and 5 present the results: while Figure 4 shows the relationship between the number of Mel Cepstral Coefficients and the classification accuracy, Figure 5 shows the relationship between the number of mel cepstral coefficients and the training time of the neural network.

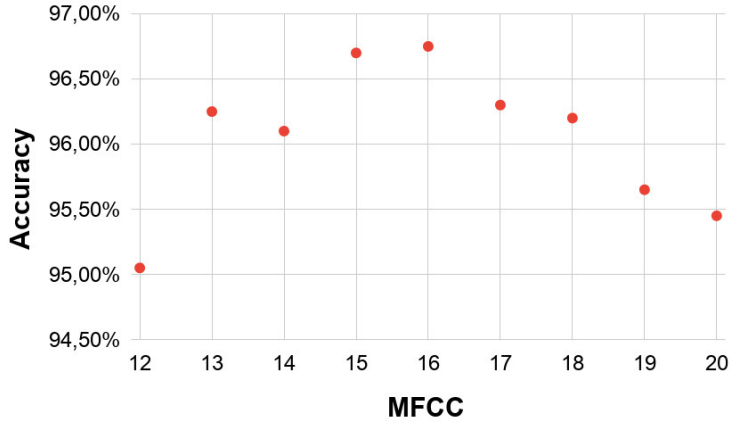


Figure 4. Classification accuracy for the first 12 to 20 *MFCC* coefficients

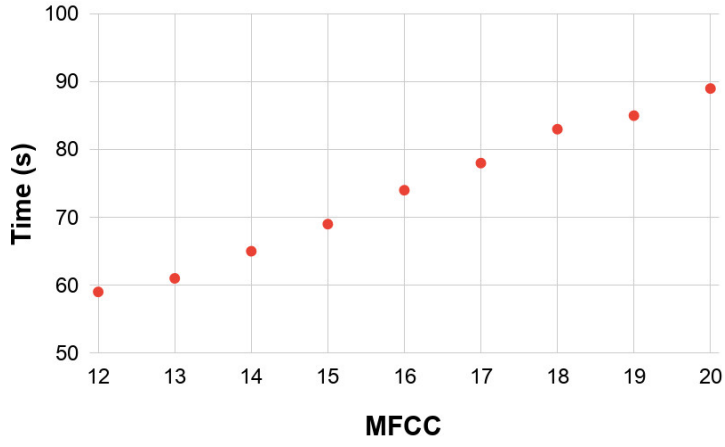


Figure 5. Training time of the neural network for the first 12 to 20 *MFCC* coefficients

The conducted experiments show that for the AudioMnist dataset, the optimal number of *MFCC* coefficients which should be taken into account is 16. Increasing the number of the coefficients has a negative effect on the training time of the neural network. Using more coefficients requires operating larger matrices and therefore requires performing more computations. We also noticed that the classification accuracy decreases when using more than 16 coefficients, what indicated that using more data would not lead to improvements in the context of classification.

5 EXPERIMENTS RESULTS

Two experiments were conducted using the modified Pytorch DNN Evolution framework. In the first one, we tried to estimate the appropriate individual size from the population of the training datasets. This experiment was aimed at evaluation of the minimum size of the training dataset for which the coevolution algorithm would still work correctly. The goal of the second experiment was to validate the correctness of the coevolutionary process and generate an architecture, which would render results comparable to one designed manually by a human researcher. The coevolution was applied to the creation of neural networks which attempted to classify sound samples.

All the test runs were performed using the same configuration. The only exception was the size of population of training datasets used in the Experiment 1, which was required due to the nature of that experiment. In all the runs 100 iterations of coevolution were performed. The parameters of the genetic algorithm are presented in Table 1.

Parameter	**Population of Neural Network**	**Population of Training Dataset**
Crossover probability	0.75	0.75
Mutation probability	0.1	0.1
Individual size	8	1 000
Population size	8	4

Table 1. Parameters of the genetic algorithm used during the coevolution

In the case of the population of neural networks, the size of the individual corresponds to the size of the generated networks. However, in the case of the population of training datasets, the size of the individual is the size of the training dataset.

5.1 Experiment 1

The first experiment using the Pytorch DNN Evolution tool was conducted to find the optimal size of the training dataset used during the evolutionary process. Such

a dataset on the one hand should be as small as possible to allow for fast individual evaluation (neural network training) and on the other hand it should contain enough samples which would allow the evolution to make progress. To find the optimal size, we conducted subsequent subexperiments in which the size of the set of training data was gradually being reduced. For each training dataset size we have conducted five runs. For a given dataset size we have recorded the maximum accuracy achieved by a neural network and the average accuracy of the best neural networks obtained through evolution but trained on a full training dataset.

Table 2 presents the accuracy for different sizes of the training dataset.

Size of the Training Dataset	**The Maximum Accuracy of Neural Networks Trained on the Subset**	**The Average Accuracy of Neural Networks Trained on the Full Dataset**
5 000	94.70 %	97.28 %
4 000	93.28 %	96.98 %
3 000	**93** %	**97.05** %
2 000	92 %	96.66 %
1 000	90 %	94.14 %
800	81 %	93.26 %
600	79 %	92.30 %
400	79 %	92.18 %
200	77 %	92.72 %
100	67 %	90.72 %

Table 2. Accuracy of training for AudioMnist subsets of different size

Based on those results, we drawn the following conclusions:

- While reducing the size of the training dataset we were obtaining lower classification accuracy in the neural networks population.
- The neural networks trained on several hundred recordings would not achieve very good accuracy in the classification. It should be noted that despite that, the effect of coevolution is still noticeable. We could observe a growing trend in the accuracy of neural network's classification for only 800 samples. The progress made by evolution in this case is presented in Figure 6.
- We have observed gains in the accuracy of the neural network when using training set sizes above 1 000 samples.
- The optimal size of the training dataset is around 3 000 samples. The accuracy of classification of the whole dataset of networks obtained through co-evolution did not grow significantly (for 4 000 it dropped to 96.98 %, for 5 000 it grew to 97.28 %) when we increased the size of training dataset further. However, the training was becoming considerably slower (Figure 7), (about 7 seconds for 4 000

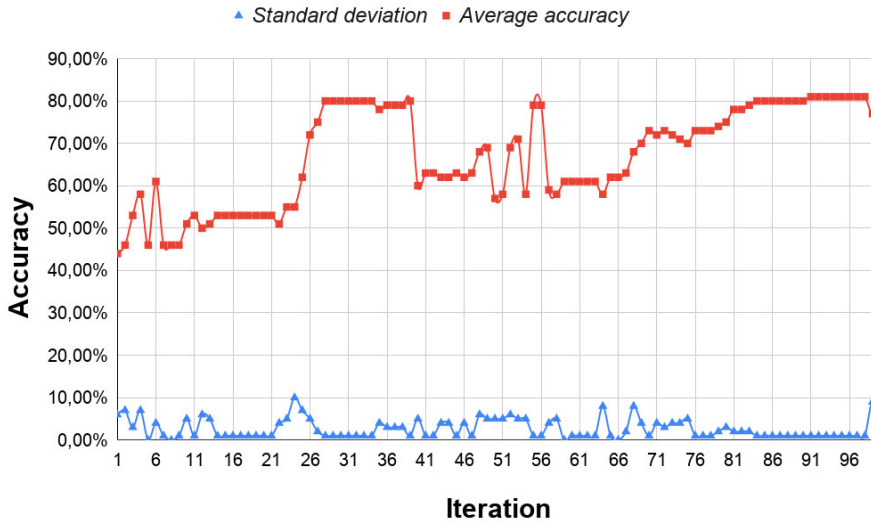


Figure 6. Accuracy of classification achieved by the population of neural networks. The size of the subset is 800. Maximum achieved accuracy is 81 %.

and about 13 seconds for 5 000) per each network training. We have decided to use the subset size of 3 000 samples in the experiments that followed.

5.2 Experiment 2

In the second experiment the goal of the coevolution process was to find a neural network architecture capable of solving the problem of digit classification. The size of the training subset was set to 3 000 recordings, as per result of Experiment 1. The graphs given in Figures 8 and 9 show the progress of the coevolution algorithm:

- Figure 8 shows the accuracy of the neural networks in the classification of digit recordings over successive iterations of the coevolutionary algorithm. The increasingly higher fitness values obtained in the subsequent iterations prove that the individuals cope better and better with speech recognition. This confirms that the evolution is able to make progress in the expected direction.
- Figure 9 shows the average accuracy of the neural network trained on individuals from the population of training datasets (fitness predictors). We can observe that on the contrary to the neural networks improving their accuracy over the course of evolution, the average accuracy of the training over the population of fitness predictors is decreasing. This suggests that in order to approximate the results of the training with a full dataset, the evolution chooses the samples, for which the classification accuracy is lowest, in other words they are hard to classify by the neural networks.

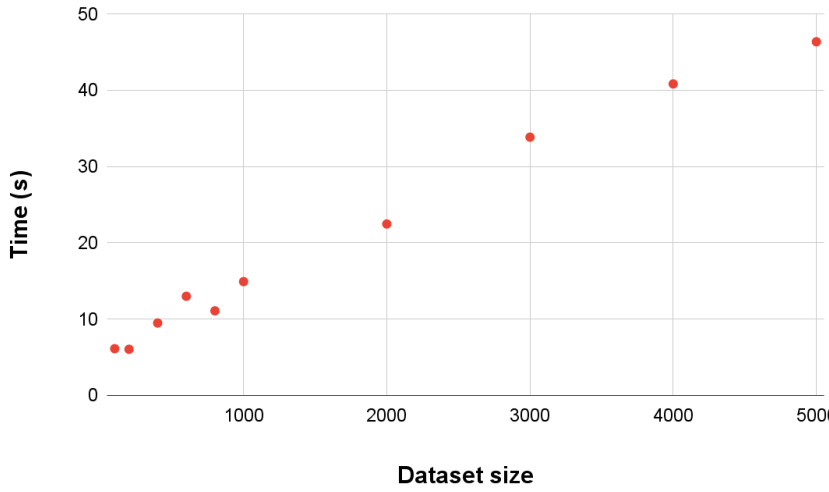


Figure 7. Training time of the neural networks for different dataset sizes

Over the course of the evolution, the maximum accuracy that was achieved was equal to 93 %. However, it should be noted that the network model in the Pytorch DNN Evolution framework was trained only on a certain subset of training data selected (the fitness predictor). Under those conditions, the classification accuracy of the neural network model is expected to be lower than in the case of training the same network model on the entire dataset. Therefore, the neural network model was trained again, however by using the entire training dataset. This allowed achieving the classification accuracy of 97.05 %.

This result can be compared, e.g. with the AlexNet model (designed by a human researcher) used in [10] which is a convolutional neural network as well. That model has also been trained to classify the AudioMnist with the use of the stochastic gradient descent and reached $95.82\,\% \pm 1.49\,\%$. In this context the result obtained by the automatically generated model could be considered satisfactory. Figure 10 presents the neural network architecture generated by Pytorch DNN Evolution. It consists of four subsequent CNN layers followed by a fully connected layer, which produces the final result (a vector of probabilities that the input sample belongs to each of the ten classes). Such a structure resembles AlexNet in which convolutional layers are also followed by the fully connected ones.

6 CONCLUSIONS

In this paper we have demonstrated how the Pytorch DNN Evolution tool can be used to automatically create a neural network architecture for the classification of

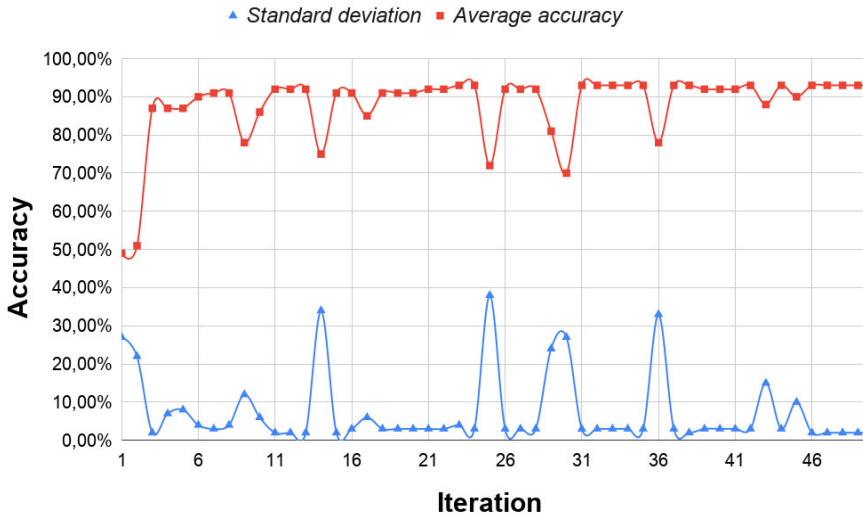


Figure 8. Classification accuracy achieved by the population of neural networks

digits in speech recordings. This approach allowed us to avoid creating the neural network architecture ourselves. Since the architecture was created with the use of an automated procedure (the coevolution process), we only had to define the elements which the network would consist of and the amount of resources we wanted to dedicate to searching for an appropriate architecture. First, in Experiment 1, we have examined what is the optimal size of the training dataset (size of an individual in the training set population). We also showed that reducing the size of individuals in the population of training subsets resulted in decreasing the neural network training time. As a consequence, the pace of the genetic algorithm accelerated. At the same time, the experiment has demonstrated that even though the size of fitness predictors was reduced, the classification accuracy did not significantly decrease. This allowed the evolutionary process to make progress towards the optimal architecture, while reducing the amount of resources required. One needs to remember, though, that trading off the accuracy for resource consumption may affect the evolution and lead it in a wrong direction. In order to avoid rendering the presented approach ineffective, it is beneficial to empirically confirm that reducing the size of fitness predictors does not lead to significant negative changes in the fitness metric values, e.g. in our case reduction of the classification accuracy. If possible the optimal size should be determined through rigorous experimentation with a wide variety of fitness predictor sizes.

In Experiment 2, the neural network obtained in the process of co-evolution achieved a classification accuracy of 97.05 %. This value is comparable to the best results achieved on the AudioMnist dataset (recognition accuracy of $95.82\,\% \pm 1.49\,\%$), described in [10]. The results obtained during the tests of the Pytorch DNN Evo-

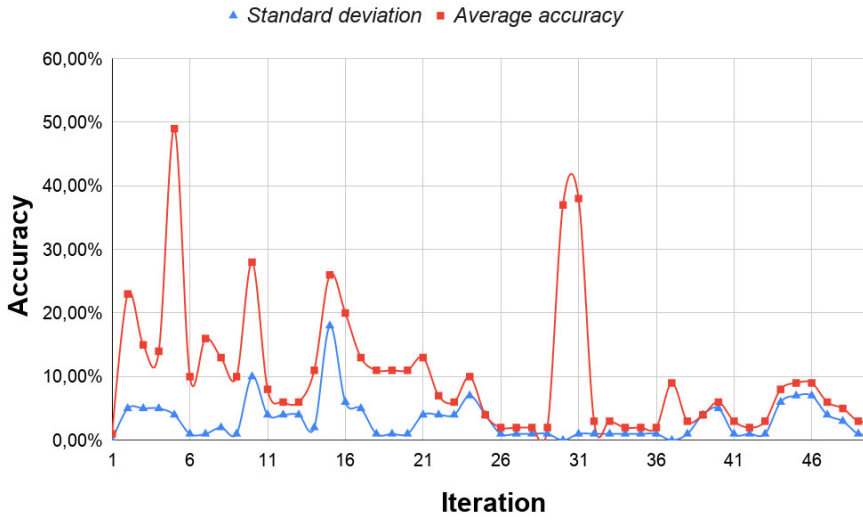


Figure 9. Classification accuracy achieved by the population of training datasets. The accuracy declines as expected: The evolution of the population of training datasets generates individuals which are harder and harder to classify correctly by the neural networks.

lution framework confirmed that coevolution can be used to search for the optimal neural network architecture, used to solve the problem of sound classification.

The positive results of the experiments prove that the data representation based on the mel cepstral coefficients is more memory-efficient than the spectrogram. The mel cepstral coefficients are approximately 10 times smaller than a spectrogram representation of the same waveform. Data complexity reduction is especially important when training neural networks as it allows to reduce the training time.

In the future, the Pytorch DNN Evolution framework can further be extended with support for other datasets. This would allow to verify whether the coevolution algorithm works also for other types of problems that may use other data types. Currently, Pytorch DNN Evolution offers the creation of neural networks by using two types of layers: convolutional layer and dense layer. We believe that extending it with new types of layers, e.g. pooling layers, recursive layers, or dropout layers, would help to apply it to more domains.

Data Availability

The datasets used, generated and analysed during the current study are available in publicly accessible repositories [5] or can be provided from the corresponding author on a reasonable request.

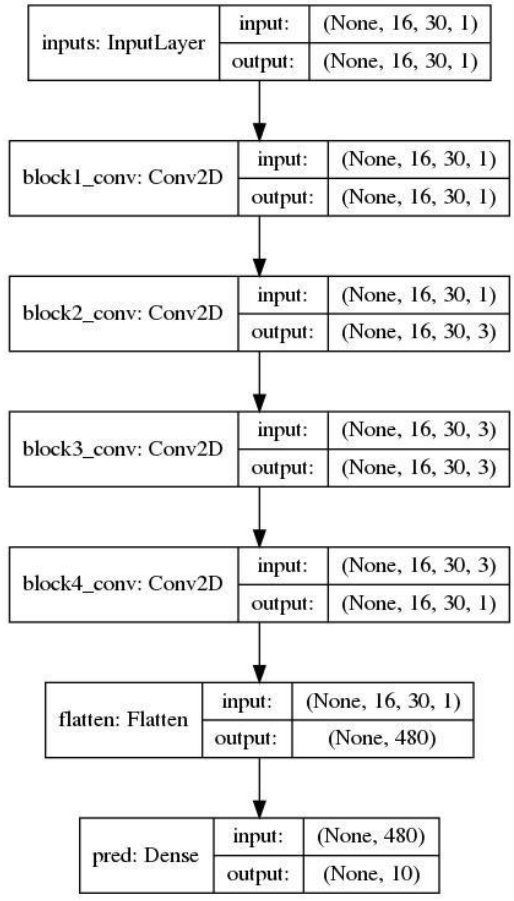


Figure 10. Neural network architecture generated by Pytorch DNN Evolution framework. The left side of the rectangle denotes the name and type the NN layer: *InputLayer* – first layer which does not transform data, *Conv2D* – convolution of input data, *Flatten* – change the format of data from a multidimensional vector to an array, *Dense* – fully connected layer. The right side specifies the format of data at the input and output to and from a given layer.

Acknowledgements

The research presented in this paper was supported by the funds assigned to AGH University of Krakow by the Polish Ministry of Education and Science. Our thanks go also to the PL-Grid infrastructure resources of the ACC CYFRONET AGH, where experiments have been carried out.

REFERENCES

[1] López, G.—Quesada, L.—Guerrero, L. A.: Alexa Vs. Siri Vs. Cortana Vs. Google Assistant: A Comparison of Speech-Based Natural User Interfaces. In: Nunes, I. L. (Ed.): Advances in Human Factors and Systems Interaction. Springer International Publishing, Cham, 2018, pp. 241–250.

[2] Tombeng, M. T.—Najoan, R.—Karel, N.: Smart Car: Digital Controlling System Using Android Smartwatch Voice Recognition. 2018 6th International Conference on Cyber and IT Service Management (CITSM), 2018, pp. 1–5, doi: 10.1109/CITSM.2018.8674359.

[3] Gundogdu, K.—Bayrakdar, S.—Yucedag, I.: Developing and Modeling of Voice Control System for Prosthetic Robot Arm in Medical Systems. Journal of King Saud University - Computer and Information Sciences, Vol. 30, 2018, No. 2, pp. 198–205, doi: https://doi.org/10.1016/j.jksuci.2017.04.005.

[4] Pytorch DNN Evolution Framework. 2018, `https://gitlab.com/pkoperek/pytorch-dnn-evolution` (Accessed: 2022-03-02).

[5] AudioMNIST Dataset. `https://github.com/soerenab/AudioMNIST` (Accessed: 2021-01-07).

[6] Niebudek-Bogusz, E.—Woznicka, E.—Korczak, I.—Śliwińska Kowalska, M.: The Applicability of Formant Voice Analysis in Diagnostics of Functional Voice Disorders. Otorynolaryngologia, Vol. 8, 2009, pp. 184–192.

[7] Mirzaei, G.—Majid, M. W.—Ross, J.—Jamali, M. M.—Gorsevski, P. V.—Frizado, J. P.—Bingman, V. P.: The BIO-Acoustic Feature Extraction and Classification of Bat Echolocation Calls. 2012 IEEE International Conference on Electro/Information Technology, 2012, pp. 1–4, doi: 10.1109/EIT.2012.6220700.

[8] Li, Z.—Peng, P.—He, Z.—Wang, L.: Automatic Classification of Microseismic Signals Based on MFCC and GMM-HMM in Underground Mines. Shock and Vibration, Vol. 2019, 2019, doi: 10.1155/2019/5803184.

[9] Bridle, J. S.—Brown, M. D.: An Experimental Automatic Word-Recognition System. JSRU Report, Vol. 1003, 1974, No. 5.

[10] Becker, S.—Ackermann, M.—Lapuschkin, S.—Müller, K.—Samek, W.: Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals. CoRR, 2018, arXiv: 1807.03418.

[11] Krizhevsky, A.—Sutskever, I.—Hinton, G. E.: Imagenet Classification with Deep Convolutional Neural Networks. Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, Curran Associates Inc., Red Hook, NY, USA, NIPS '12, 2012, pp. 1097–1105.

[12] Zhang, B.—Leitner, J.—Thornton, S.: Audio Recognition Using Mel Spectrograms and Convolution Neural Networks. Technical Report, 2019.

[13] Simonyan, K.—Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014, arXiv: 1409.1556.

[14] Lecun, Y.—Bottou, L.—Bengio, Y.—Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, Vol. 86, 1998, No. 11, pp. 2278–2324, doi: 10.1109/5.726791.

[15] Guiming, D.—Xia, W.—Guangyan, W.—Yan, Z.—Dan, L.: Speech Recognition Based on Convolutional Neural Networks. 2016 IEEE International Conference on Signal and Image Processing (ICSIP), 2016, pp. 708–711, doi: 10.1109/SIPROCESS.2016.7888355.

[16] Huang, J. T.—Li, J.—Gong, Y.: An Analysis of Convolutional Neural Networks for Speech Recognition. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 4989–4993, doi: 10.1109/ICASSP.2015.7178920.

[17] Rumelhart, D. E.—Hinton, G. E.—Williams, R. J.: Learning Representations by Back-Propagating Errors. Nature, Vol. 323, 1986, No. 6088, pp. 533–536, doi: 10.1038/323533a0.

[18] Hochreiter, S.—Schmidhuber, J.: Long Short-Term Memory. Neural Computation, Vol. 9, 1997, No. 8, pp. 1735–1780.

[19] Vaswani, A.—Shazeer, N.—Parmar, N.—Uszkoreit, J.—Jones, L.—Gomez, A. N.—Kaiser, L.—Polosukhin, I.: Attention Is All You Need. CoRR, 2017, arXiv: 1706.03762.

[20] Chen, M. X.—Firat, O.—Bapna, A.—Johnson, M.—Macherey, W.—Foster, G.—Jones, L.—Schuster, M.—Shazeer, N.—Parmar, N.—Vaswani, A.—Uszkoreit, J.—Kaiser, L.—Chen, Z.—Wu, Y.—Hughes, M.: The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Melbourne, Australia, 2018, pp. 76–86, doi: 10.18653/v1/P18-1008.

[21] Wu, Y.—Schuster, M.—Chen, Z.—Le, Q. V.—Norouzi, M.—Macherey, W.—Krikun, M.—Cao, Y.—Gao, Q.—Macherey, K.—Klingner, J.—Shah, A.—Johnson, M.—Liu, X.—Kaiser, L.—Gouws, S.—Kato, Y.—Kudo, T.—Kazawa, H.—Stevens, K.—Kurian, G.—Patil, N.—Wang, W.—Young, C.—Smith, J.—Riesa, J.—Rudnick, A.—Vinyals, O.—Corrado, G.—Hughes, M.—Dean, J.: Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation. CoRR, 2016, arXiv: 1609.08144.

[22] Ren, P.—Xiao, Y.—Chang, X.—Huang, P.—Li, Z.—Chen, X.—Wang, X.: A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. CoRR, 2020, arXiv: 2006.02903.

[23] Chitty-Venkata, K. T.—Emani, M.—Vishwanath, V.—Somani, A. K.: Neural Architecture Search for Transformers: A Survey. IEEE Access, Vol. 10, 2022, pp. 108374–108412, doi: 10.1109/ACCESS.2022.3212767.

[24] Cui, J.—Chen, P.—Li, R.—Liu, S.—Shen, X.—Jia, J.: Fast and Practical Neural Architecture Search. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 6508–6517, doi: 10.1109/ICCV.2019.00661.

[25] Cai, H.—Zhu, L.—Han, S.: ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. International Conference on Learning Representations, 2019, doi: 10.48550/arXiv.1812.00332.

[26] Zela, A.—Elsken, T.—Saikia, T.—Marrakchi, Y.—Brox, T.—Hutter, F.:

Understanding and Robustifying Differentiable Architecture Search. CoRR, 2019, arXiv: 1909.09656.

[27] Gao, J.—Xu, H.—Shi, H.—Ren, X.—Yu, P. L. H.—Liang, X.—Jiang, X.—Li, Z.: AutoBERT-Zero: Evolving BERT Backbone from Scratch. CoRR, 2021, arXiv: 2107.07445.

[28] Wistuba, M.: Deep Learning Architecture Search by Neuro-Cell-Based Evolution with Function-Preserving Mutations. In: Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., Ifrim, G. (Eds.): Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, Cham, 2019, pp. 243–258.

[29] Real, E.—Aggarwal, A.—Huang, Y.—Le, Q. V.: Regularized Evolution for Image Classifier Architecture Search. CoRR, 2018, arXiv: 1802.01548.

[30] Guo, Z.—Zhang, X.—Mu, H.—Heng, W.—Liu, Z.—Wei, Y.—Sun, J.: Single Path One-Shot Neural Architecture Search with Uniform Sampling. CoRR, 2019, arXiv: 1904.00420.

[31] Funika, W.—Koperek, P.: Co-Evolution of Fitness Predictors And deep Neural Networks. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics. Springer International Publishing, Cham, 2018, pp. 555–564.

[32] LeCun, Y.—Cortes, C.: MNIST Handwritten Digit Database. 2010, http://yann.lecun.com/exdb/mnist/.

[33] Natsiou, A.—O'Leary, S.: Audio Representations for Deep Learning in Sound Synthesis: A Review. CoRR, 2022, arXiv: 2201.02490.

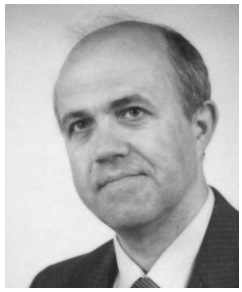
[34] Laguarta, J.—Hueto, F.—Subirana, B.: COVID-19 Artificial Intelligence Diagnosis Using Only Cough Recordings. IEEE Open Journal of Engineering in Medicine and Biology, Vol. 1, 2020, pp. 275–281, doi: 10.1109/OJEMB.2020.3026928.

[35] Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Technical Report, 2009, pp. 32–33, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[36] Documentation of Torchaudio Library. (Accessed: 2021-01-07).

[37] Qu, X.—Wei, P.—Gao, M.—Sun, Z.—Ong, Y. S.—Ma, Z.: Synthesising Audio Adversarial Examples for Automatic Speech Recognition. Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, KDD '22, 2022, pp. 1430–1440, doi: 10.1145/3534678.3539268.

[38] Chen, G.—Zhao, Z.—Song, F.—Chen, S.—Fan, L.—Wang, F.—Wang, J.: Towards Understanding and Mitigating Audio Adversarial Examples for Speaker Recognition. 2022, doi: 10.48550/ARXIV.2206.03393.



Włodzimierz Funika works at the Institute of Computer Science of the AGH University in Krakow (Poland). His main research interests are in distributed computing, tool construction, performance analysis and visualization, data science, and machine learning. Involved in many EU-funded projects and Polish-wide projects: PL-Grid and others.

Pawel Koperek is a researcher in the field of machine learning. He received his Master of Science degree in computer science in 2010 from the AGH University of Science and Technology, where he studied at the Faculty of Electrical Engineering, Automatics, Computer Science and Electronics. He has a passion for exploring new ideas in machine learning and is particularly interested in evolutionary algorithms, deep learning and deep reinforcement learning and their practical applications (e.g., in the automatic cloud resource management domain).

Tomasz Wiewióra graduated with his Master's in the field of computer science at the Faculty of Computer Science, Electronics and Telecommunications of the AGH University of Science and Technology in Krakow. He works as a programmer in a large company, where he deals with the implementation of solutions for clients from the banking industry.