ISSN 1870-4069

Tuning the Parameters of a Convolutional Artificial Neural Network by Using Covering Arrays

Humberto Pérez-Espinosa[1,2], Himer Avila-George[1,2], Josefina Rodriguez-Jacobo[1], Hector A. Cruz-Mendoza[1], Juan Martínez-Miranda[1,2], Ismael Espinosa-Curiel[1]

[1] CICESE-UT^3, Tepic, Nay., Mexico

[2] CONACYT - CICESE-UT^3, Tepic, Nay., Mexico

hperez@cicese.mx, himerag@cicese.mx
http://idi.cicese.mx/ut3/

Abstract. Artificial Neural Networks have proven to be a very powerful machine learning algorithm which can be adequate to learn successfully a variety of tasks. Currently, very complex classification problems on different kind of data (images, video, sound, text, DNA) have been solved using neural networks. This kind of algorithms usually has many parameters that need to be fine-tuned in order to have good results. Usually, this tuning is made by trial and error. However, this procedure does not guarantee the optimal performance of the training process. In this work, we study the use of mixed-level covering arrays to design experiments that help us to find the best combinations of parameters. We tested this approach by tuning a convolutional neural network for an audio classification task. For the implementation, we took advantage of the flexibility of the open source software library for machine learning TensorFlow.

Keywords: Artificial neural networks; covering arrays; artificial neural networks tuning; ConvNet architectures.

1 Introduction

Currently, artificial neural networks are used for solving a variety of machine learning tasks. The use of deep architectures has allowed going beyond the limits reached with other architectures in certain tasks such as speech recognition and image classification. This new revival of neural networks has even resulted in the successful creation of mass consumption products and services with a very high added value. For example, products for the classification of image, video, text, written language, speech and audio. Services for product recommendation, relational data and social networks mining. Deep learning methods aim at learning feature hierarchies. The features from higher levels of the hierarchy are formed by the composition of lower lever features [4]. Despite the great potential of deep neural networks, it is not easy to tune them. The problem of identifying the parameters for a specific structure of a neural network is an important research topic.

There have been several research efforts focused on the tuning of artificial neural networks (ANN). Some researchers have applied genetic algorithms for tuning the parameters of the network[3] and some others for tuning the parameters and also the structure of the network [13] [9]. Bashiri *et al.* [3] used central composite to design experiments and also to analyze the behavior of the neural network according to some identified parameters by using an overall desirability function. The genetic algorithm was applied to find optimal parameter status. They varied three parameters to evaluate the performance of the ANN: the percentage of trained data, the number of neuron in the first layer and the number of neuron in the second layer. Central composite was applied to extract the relationship between responses and controllable factors. To validate the proposed method, they compared their results versus the obtained applying the Taguchi methodology. Taguchi methodology is considered an important tool for robust design, i.e. for optimizing the product of process conditions which are minimally sensitive to the various causes of variation. This methodology has been used in several works for tuning ANN. In the work done by Tsai *et al.* [13] the authors used a hybrid Taguchi-genetic algorithm to tune the network structure and also the parameters of a feed-forward neural network. The numbers of hidden nodes and the links are chosen by increasing them from small numbers until the learning performance is good enough. A similar approach was proposed by Leung *et al.* [9], the method tunes the structure and parameters of an ANN by using an improved genetic algorithm. The neural network is able to learn both, the input-output relationships of an application and the network structure using the improved genetic algorithm. The authors propose the use of an ANN with link switches and its particularity is a unit step function introduced to each link. They observed that the weights of the links govern the input-output relationship of the ANN, while the switches of the links govern its structure. Tortum *et al.* [12] also applied the Taguchi method in the optimization of the design parameters of an ANN.

Besides genetic algorithms, other evolutionary approaches have been tested to solve the ANN tuning problem. Xiao *et al.* [14] applied a good point set-evolutionary strategy for tuning both, the network structure and parameters of a feed-forward neural network. The good point set is incorporated to enhance the crossover operator of the evolutionary strategy. The numbers of hidden nodes and the links of the feed-forward neural networks were chosen by increasing from small number of both until the learning performance is good enough. Zhao *et al.* [15] applied a cooperative binary-real particle swarm optimization to find compact structures and optimal parameters of an ANN with link switches which define the structure of the network. The works found in the literature have several points in common:

- The optimization of feed-forward neural networks [14], [15], [13], [9],
- The use of evolutionary strategies [14], [15], [13], [9], [3],
- The optimization of the number of hidden nodes and links [14], [15], [13], [9],
- The structure of the ANN is chosen by increasing the number of nodes and links until the learning performance is good enough [14], [15], [13], [9],
- The combination of an optimization technique with an experiment design method [14], [13], [3], [12].

In recent years, some combinatorial objects called covering arrays (CAs) have become very important in the design of experiments [7,8,10]. CAs are a generalization of orthogonal arrays (OAs). In the 80s, G. Taguchi [11] promoted the use of OAs as templates for developing experimental plans. The problem is that not OAs exist for certain configurations of variables, levels, and interaction degrees. Therefore, it is often difficult to choose a suitable OA to a specific problem. A CA is a combinatorial object that does not need to be balanced (not all t-tuples need to appear the same number of times) to open the possibility of create CAs with different configurations of variables, levels, and interaction degrees. A mixed-level covering array denoted by $MCA(N; t, k, (v_1, \ldots, v_k))$ is a $N \times k$ matrix in which the entries of the ith column arise from an alphabet of size v_i; additionally, each column $i(1 \leq i \leq k)$ contains only elements from a set S_i with $|S_i| = v_i$, and the rows of each $N \times t$ subarray cover all t-tuples of values from the t columns at least once.

In this work, we propose the use of an experiment design method, based on mixed-level covering arrays (MCA), for tuning parameters of a Convolutional Neural Network (ConvNet). The characteristics of our approach, in contrast with the related works found in the literature, are:

- Optimization of a deep architecture neural network (ConvNet),
- The use of an experimental design instead of evolutionary strategies,
- The tuning of a relatively large number of parameters, some for tuning neural networks in general and some for tuning ConvNet architectures in particular,
- The structure of the ANN (number and type of layers) is fixed. We only change the parametrization,
- We do not apply any optimization technique.

The experiments presented in this work are based on two research questions:

1. Are mixed-level covering arrays a good option to fine tune the performance of an ANN?
2. How much significant is the improvement obtained using this approach?

2 Methodology

In order to answer the research questions posed in the previous section, we conducted an experiment in which we implemented a convolutional neural network using an open source tool. We identified the set of parameters that need to be tuned and determined their possible range of values. Using MCA, we obtained a list of parameter combinations to test. We run the testing cases on an audio classification task. In the following subsections, we explain the details of the proposed method.

2.1 TensorFlow

For the implementation of the convolutional neural network, we used the software TensorFlow [1]. It was originally developed by researchers and engineers from the Google's Brain Team with the objective of conducting machine learning and deep neural networks research. This software is currently used for both, research and production by different teams in several of the commercial Google products, such as speech recognition, Gmail, Google Photos, and Search. An important characteristic of this tool is its flexible architecture which allows deploying computation to one or more CPUs or GPUs in a desktop, server, or mobile device using the same API. Google released TensorFlow under the Apache 2.0 open source license in November 2015.

2.2 ConvNets

ConvNets are a type of neural networks inspired by the visual system's structure. In these networks, the weights are shared across time or space. Neurons with the same weights are applied on input patches of the previous layer at different segments of the input data. In this way, it is obtained a *translational invariance* that allows the networks learning patterns and reuse them on different space or time context. Therefore, ConvNets are useful when the inputs samples are statistical invariants, that is, the input samples contain the same kind of information and it does not change on average across time or space. In a ConvNet, instead of having stacks of matrix multiply layers, there are stacks of convolutions. When the size of the patch is the same size than the whole feature vector, it is just a regular layer of a neural network. But when the patch is smaller than the whole feature vector, there are fewer weights and they are shared along the sequence of input features. Currently, pattern recognition systems based on convolutional networks are among the best performing systems. This type of network architecture typically have five, six or seven layers, a number of layers which makes fully connected neural networks almost impossible to train properly when initialized randomly [4].

2.3 ConvNet Structure

ConvNets are usually structured by stacking up convolution layers. At the top of the structure, there are fully connected layers. And finally, there is a classifier. Stride is the number of features that are shifted each time the filter moves. The strides between layers are used to reduce the dimensionality and to increase the depth of the neural network. A stride of one makes the output the same size as the input. A stride of 2 means that output is half the size as the input. Each layer in the structure is called a feature map. There could be more than one feature map, for instance, if an image has the channels R,G, and B, these three channels (K-size = 3) can be handled as individual features maps. In our particular case, we are handling only one channel because the audio recording are mono-channel. When the shifting filter does not go beyond the edge of the feature vector, it is called valid padding. If the filter goes off the edge of the feature vector and therefore the output map size is exactly the same size than the input map, it is called same padding.

One possible improvement to this configuration of a ConvNet is to reduce the extent of the feature maps in the convolutional stack. Striding to shift the filters by a few features each time removes important information and produce less accurate models. The pooling layers take all the convolutions in a neighborhood and combine them instead of skipping one in every two convolutions. Max pooling take a small neighborhood around every point in the feature map and compute the maximum of all the responses around it. Given that the convolutions are done on lower stride, the structure becomes more expensive to compute and there are more parameters to tune. Average pooling instead of taking the max, just take an average over the window of features around a specific location.

For our implementation, we are using a typical architecture for ConvNet. It has two alternated layers of convolution and pooling, followed by a fully connected layer and a classification layer at the end. The Fig. 1 shows the network structure.

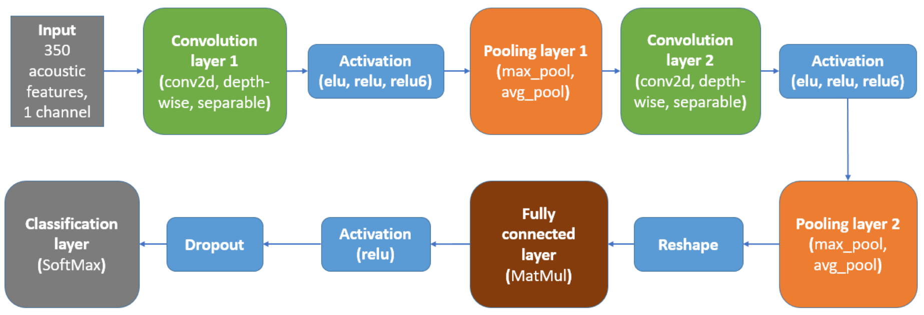
Input 350 acoustic features, 1 channel
Convolution layer 1 (conv2d, depth-wise, separable)
Activation (elu, relu, relu6)
Pooling layer 1 (max_pool, avg_pool)
Convolution layer 2 (conv2d, depth-wise, separable)
Activation (elu, relu, relu6)
Pooling layer 2 (max_pool, avg_pool)
Reshape
Fully connected layer (MatMul)
Activation (relu)
Dropout
Classification layer (SoftMax)

Fig. 1: Convolutional neural network architecture used for the experiments presented in this paper. The different options that we tested at each layer are shown in parentheses.

2.4 ConvNet Parametrization

We identified three types of parameters. General: that are common in the training of any artificial neural network (e.g., number of epochs, learning rate).

Convolutional: that are particular parameters of a basic ConvNet (e.g., the stride to slide the filter, type of padding). Improvement: that are parameters of operations used to improve the performance of a ConvNet (e.g., pooling region size, pooling stride).

General:

v1 - **Num_epochs:** a positive integer that limits the number of steps that validation set is evaluated.
v2 - **Learning_rate:** a float that represents the magnitude of the update per each training step. Decay once per epoch.
v3 - **Batch_size:** a positive integer that indicates the percentage of the training data that are feed in each iteration.
v4 - **Eval_batch_size:** a positive integer that indicates the percentage of the validation data that is feed in each iteration.
v5 - **Loss:** Operation to measure the error of a network. The possible values for the parameter are reduce_max and reduce_min.
v6 - **Optimizer:** functions to compute gradients for a loss measure and apply gradients to variables. The four possible values of this parameter are *MomentumOptimizer*, *GradientDescentOptimizer*, *AdamOptimizer* and *RMSPropOptimizer*.
v7 - **Activation:** functions that provide nonlinearities. This parameter takes three possible values. One function for smooth nonlinearities (*elu*) and two continuous but not everywhere differentiable functions (*relu*, *relu6*).
v8 - **Classification:** operations to perform classification. In our case, *softmax* activations.

Convolutional:

v9 - **K-size** an integer that represents the size of the dimension of the input.
v10 - **Strides** a list of integers that represents the number of features that are shifted by the sliding window for each dimension of the input vector each time the filter moves.
v11 - **Convolution** functions to sweep a filter over a batch of input data, applying the filter to each window of each input vector of the appropriate size. This parameter takes three possible values *conv2d*, *depthwise_conv2d* and *separable_conv2d*.
v12 - **Padding** If the filter goes off the edge of the feature map or not. The two possible values for this parameter are same and valid.
v13 - **Patch_size** a positive integer that indicates the size of the window that slides across the feature vector. It could be also bidimensional, in our case it is unidimensional.
v14 - **Depth1** a positive integer that represents the depth of the first convolution layer.

v15 - Depth2 a positive integer that represents the depth of the second convolution layer.

Improvement:

v16 - Pool function to reduce the extent of the feature maps sweeping a rectangular window over the input. The two possible values for this parameters are *max_pool* and *avg_pool*.

In Table 1**??**, the sixteen main variables and their levels/values are shown.

2.5 Criteria to Evaluate the ConvNet Parametrization

The criteria that we used to select the best parametrization of the ConvNet was a measurement of the classification performance. The results tables show the performance of the classification experiments in terms of precision, recall and F-measure. The F-measure is a classification performance metric that is calculated as the harmonic mean of precision and recall. The F-measure score reaches its best value at 1 when precision and recall are 1, and worst at 0 when precision or recall is 0. This is, the closest to 1, the most accurate is the classification. To measure the performance we use three data subsets: training, validation, and testing. The results showed in the tables are the ones obtained with the test set.

2.6 Design of Experiments based on MCA

The methodology for setting the values of the parameters of an ANN is based on the study of the effect on the quality of the solution, which is caused by the interaction between the variables of the experiment. The tuning process of the parameters of the ANN was done using an $MCA(49; 2, 7^4 5^1 4^3 3^4 2^3)$, this MCA represents the experimental plan. The MCA was constructed using the simulated annealing algorithm reported in [2]. The experimental plan is composed of fifty-two experiments, see Table 1**??**. In the construction of the experimental design, only fifteen parameters were used, because the parameter *Classification* only has one level.

2.7 Used Data

To find out which is the best configuration of the ConvNet, we used data from a dog barks database created by the Haramara TIC Lab at CICESE-UT[3] in Nayarit, Mexico. The recorded barks of this database were induced applying four stimuli, three of them were related to negative emotions (aggression to unfamiliar people), and the fourth with a positive emotion (playfulness). The dogs were recorded in their habitual environment.

Aggression to unfamiliar people:
We divided this procedure into three parts:

Table 1: The design of experiment based on a MCA. **??** Variables and their levels. **??** The experimental plan.

(a)

	Variables															
Levels	***v1***	***v2***	***v3***	***v4***	***v5***	***v6***	***v7***	***v8***	***v9***	***v10***	***v11***	***v12***	***v13***	***v14***	***v15***	***v16***
0	20	0.0001	8	8	tf.reduce_max	Momentum	relu	softmax	1	1	tf.nn	same	3	8	32	max
1	30	0.0006	20	20	tf.reduce_mean	GradientDescent	relu6		2	2	tf.nn.depthwise	valid	4	16	64	avg
2	40	0.0011	32	32		Adam	elu		3	3	tf.nn.separable		5	32	96	
3	50	0.0016	44	44		RMSProp							6	64	128	
4	60	0.0020	56	56									7			
5	70	0.0025	68	68												
6	80	0.0030	80	80												

(b)

Experiment #	***Variables***														
	v1	***v2***	***v3***	***v4***	***v5***	***v6***	***v7***	***v9***	***v10***	***v11***	***v12***	***v13***	***v14***	***v15***	***v16***
exp01	3	3	3	5	0	0	1	0	1	1	0	1	1	0	1
exp02	1	1	1	0	0	1	1	1	1	0	0	1	1	3	1
exp03	1	1	1	5	0	2	0	0	2	1	0	3	1	1	1
exp04	2	2	2	4	1	0	0	2	0	0	1	1	0	1	0
exp05	0	0	0	1	1	0	1	1	1	1	0	0	0	1	0
exp06	0	0	0	5	0	1	2	0	2	2	1	4	1	1	1
exp07	4	4	4	6	0	2	0	2	0	1	0	1	0	3	0
exp08	1	1	1	4	1	0	1	1	1	0	1	2	3	2	1
exp09	2	2	2	0	1	3	2	2	2	0	1	0	0	3	0
exp10	4	4	4	3	0	0	0	0	0	0	0	4	1	2	1
exp11	4	4	4	1	0	1	1	1	1	1	0	1	2	1	0
exp12	5	5	5	3	1	3	2	2	2	2	1	1	3	0	1
exp13	5	5	5	5	0	3	0	1	1	0	0	0	3	3	0
exp14	2	2	2	6	1	3	1	2	0	2	1	4	2	1	1
exp15	0	0	0	2	1	3	2	2	1	2	0	3	0	2	1
exp16	4	4	4	2	0	3	0	2	0	2	1	0	1	0	0
exp17	2	2	2	3	0	0	1	1	1	0	0	3	0	0	0
exp18	0	0	0	3	1	1	2	0	2	1	1	2	2	3	1
exp19	6	6	6	4	0	2	0	0	0	0	0	4	2	2	1
exp20	4	4	4	0	1	3	1	1	1	2	1	4	3	3	0
exp21	5	5	5	6	0	1	2	2	2	2	1	2	0	2	0
exp22	0	0	0	4	1	0	0	0	0	0	0	1	1	2	1
exp23	3	3	3	2	0	2	1	1	1	1	0	4	0	2	0
exp24	1	1	1	2	1	2	2	2	2	2	1	1	3	0	1
exp25	6	6	6	2	0	2	0	0	0	0	0	1	2	0	0
exp26	1	1	1	5	1	1	1	1	1	1	1	2	0	0	1
exp27	5	5	5	1	0	2	2	2	2	2	0	4	0	3	0
exp28	1	1	1	6	1	3	2	0	1	0	1	0	0	1	1
exp29	0	0	0	6	0	0	1	1	0	2	1	4	1	0	0
exp30	6	6	6	3	1	0	2	2	2	1	0	1	1	3	1
exp31	1	1	1	3	0	2	0	2	0	2	0	0	2	2	0
exp32	6	6	6	1	1	3	1	2	1	1	1	2	1	3	1
exp33	4	4	4	4	0	3	2	1	2	1	0	3	0	0	1
exp34	6	6	6	6	1	1	0	0	0	1	1	3	3	1	0
exp35	2	2	2	1	0	2	1	1	2	1	0	0	3	2	0
exp36	5	5	5	4	1	0	2	0	1	2	1	3	2	3	1
exp37	1	1	1	1	0	1	0	0	2	0	0	4	3	0	0
exp38	6	6	6	1	1	1	1	0	0	1	1	3	2	1	1
exp39	3	3	3	6	1	3	2	2	0	0	1	3	1	2	0
exp40	3	3	3	4	0	1	0	0	1	0	0	0	3	1	1
exp41	5	5	5	0	0	2	1	1	0	1	0	3	0	0	0
exp42	6	6	6	5	1	0	2	2	1	2	1	0	1	2	1
exp43	3	3	3	0	1	2	0	2	0	1	1	2	2	2	1
exp44	0	0	0	0	0	2	1	0	1	0	0	0	3	1	0
exp45	6	6	6	0	0	0	2	1	2	2	1	3	0	0	1
exp46	3	3	3	1	1	0	0	0	2	1	0	1	3	3	0
exp47	2	2	2	5	0	1	1	1	0	0	0	2	1	2	0
exp48	5	5	5	2	1	1	2	2	2	0	1	2	1	1	1
exp49	3	3	3	3	0	3	0	0	0	2	0	3	2	1	0
exp50	4	4	4	5	1	1	0	1	0	1	1	2	2	1	1
exp51	1	1	1	0	0	2	2	2	2	2	0	4	2	2	0
exp52	2	2	2	2	1	0	0	0	0	0	1	1	2	3	1

- Aggr1: The dog started barking when it realized the presence of a stranger at the door. After the door was open, the experimenter stimulated aggressive barks by doing some threatening movements in front of the dog and kept recording until the dog stop barking.
- Aggr2: The experimenter moved away from the owner house for ten minutes and then returned. The experimenter pretended to be a thief, hit the door and tried to force it to open. In this case, the barks were more intense.
- Aggr3: Again the experimenter moved away from the owner house and then

returned and he looked out through the window or above the wall of the house.

Playfulness:
The experimenter asked the owner to play with the dog using a toy or its favorite object. The experimenter, then, recorded the interaction between the owner and the dog.

From these inductions we obtained four categories of barks that we will try to classify using a neural network:

- **Aggressive-bark:** bark occurred due to the presence of a stranger.
- **Very-Aggressive-bark:** bark occurred due to the presence of a stranger and imminent threat.
- **Play-bark:** bark occurred while the dog is playing.
- **Other-bark:** bark that was not produced by some of the planned stimuli.

2.8 Audio Characterization

We characterized acoustically the audio data from this database using the software openSMILE [5]. This software allowed us to extract the following Low-Level Descriptors (LLDs): Melspec, MFCC, Energy, Spectral Bands, Spectral Roll Off, Spectral flux, Spectral centroid, Spectral MaxPos, Spectral MinPos, Voice prob, F0Env, F0 envelope, F0 and zero crossing rate. We computed these acoustic features using a frame size of 25 ms and a frame step of 10 ms. We applied a moving average filter for smoothing data contours. We calculated the delta and double delta regression coefficients for the values of LLDS in each frame. We calculated 39 statistical functions over the values of the LLDs, its deltas, and its double deltas coefficients in each frame of the sample.

We obtained a total of 6,552 attributes for each single audio sample. After an experimentation stage we decided to use the *Relief Attribute* evaluation method as implemented in Weka [6]. The method showed the best accuracy rates when we took the 350 best-ranked attributes. We selected these features from the original feature set of 6,552 attributes to obtain the best attributes and reduce the dimensionality of the attributes vector.

3 Results

For the experiment implementation, we used the TensorFlow framework installed in a Linux server called Harvest. Harvest is part of the computing infrastructure of the Center for Scientific Research and Higher Education at Ensenada, Technology Transfer Unit at Tepic (CICESE-UT3). Harvest is equipped with 72 Intel Xeon processors, 64GB RAM, and a Tesla K20 GPU accelerator. The Linux distribution for this server is CentOS.

Fifty-two experiments were run (see Table 2). We observed a diversity of classification performances. From very bad results (F-Measure below 0.30) to very good results (F-Measure above 0.90). We identified some parameters that have a more evident effect on the results, for example, learning rate, convolution, and batch size. Other parameters do not have an evident effect on the results, for example, patch size, depth, and activation. We found that the best configuration is **exp39**, this experiment reaches a *F-Measure = 0.94*.

Three combinations were not possible because when the convolution is a *depth_wise* the variable depth must be the divisor of *num_hidden*. And when the convolution is separable the variable depth must be less than *num_hidden*. Those three cases did not satisfy this condition (*exp12*, *exp24* and *exp35*).

Table 2: Experimental design results.

Experiment #	***Results***		
	Precision	***Recall***	***F-Measure***
exp01	0.32	0.57	0.41
exp02	0.62	0.63	0.59
exp03	0.55	0.58	0.55
exp04	0.78	0.76	0.73
exp05	0.32	0.57	0.41
exp06	0.45	0.56	0.45
exp07	0.00	0.06	0.01
exp08	0.73	0.72	0.70
exp09	0.90	0.90	0.90
exp10	0.79	0.79	0.79
exp11	0.47	0.52	0.49
exp12			
exp13	0.71	0.61	0.61
exp14	0.32	0.57	0.41
exp15	0.00	0.05	0.00
exp16	0.59	0.32	0.16
exp17	0.57	0.58	0.56
exp18	0.32	0.57	0.41
exp19	0.74	0.70	0.70
exp20	0.32	0.57	0.41
exp21	0.32	0.57	0.41
exp22	0.32	0.57	0.41
exp23	0.16	0.05	0.01
exp24			
exp25	0.92	0.92	0.92
exp26	0.32	0.57	0.41
exp27	0.10	0.32	0.16
exp28	0.88	0.88	0.87
exp29	0.10	0.32	0.16
exp30	0.32	0.57	0.41
exp31	0.10	0.32	0.16
exp32	0.32	0.57	0.41
exp33	0.44	0.57	0.41
exp34	0.32	0.57	0.41
exp35			
exp36	0.32	0.57	0.41
exp37	0.59	0.58	0.58
exp38	0.32	0.57	0.41
exp39	0.94	0.94	0.94
exp40	0.67	0.67	0.67
exp41	0.40	0.41	0.37
exp42	0.32	0.57	0.41
exp43	0.34	0.56	0.43
exp44	0.74	0.54	0.50
exp45	0.32	0.57	0.41
exp46	0.32	0.57	0.41
exp47	0.60	0.60	0.60
exp48	0.32	0.57	0.41
exp49	0.32	0.57	0.41
exp50	0.32	0.57	0.41
exp51	0.00	0.06	0.01
exp52	0.86	0.86	0.86

The Table 3 shows the classification performance metrics of the best parameters combination obtained by the MCA. This result is compared with the classification results obtained with some widely used machine learning algorithms. We used the software Weka [6] to run the training algorithms. We used the default parametrization of this tool for all the algorithms. The evaluation was done using the same data subsets for all the classifiers. The samples were randomly distributed in the three subsets, 2,246 samples in the testing set, 642 in the validation set and 321 samples in the test set. Features vectors consist of 350 acoustic features.

We observed a marked difference in the classification performance between the ConvNet with tuned parametrization and the rest of classifiers. This comparison shows that by applying the proposed method for parametrization of ConvNets, we can obtain very accurate classification models. Random Forest and Support Vector Machines showed good results (*F-Measure* = 0.79 and 0.78 respectively) but there is a significant difference in the results showed by the ConvNet (0.94). Eight of the configurations obtained an *F-Measure* above the results obtained with Random Forest and Support Vector Machines.

Table 3: Comparison between the best result of the experiment and other classifiers.

Classifier	*Precision*	*Recall*	*F-measure*
Tuned ConvNet	0.94	0.94	0.94
Naive Bayes	0.70	0.67	0.68
Support Vector Machines	0.78	0.79	0.78
C4.5	0.69	0.69	0.69
Random Forest	0.79	0.79	0.78

4 Conclusions and Future Work

In this work, we applied MCA to design an experiment with the objective of finding good combinations of parameters for a ConvNet. We found out that running the experiments resulted in a variation of classification performance from very bad to very good. Given the best classification result from the experiment, we can conclude that this method for tuning the parameters of a ConvNet is a good option. We compared the best result obtained with the proposed method against the results obtained with other classifiers. We observed a significant improvement in relation with the other classifiers.

The results obtained in this work are very encouraging to keep exploring the use of MCA for tuning deep ANN architectures. As future work, we are planning to do a more exhaustive and formal evaluation of the classification performance and benefits of the proposed method. In that evaluation, we plan to test the following aspects:

– Testing of other audio, image and text databases.
– Testing with regression tasks.
– Testing of other deep structure variants such as auto-encoders, inceptions, and long short-term memory.
– Comparison with other ANN fine-tuning methods.
– Tuning of the structure of the deep architecture number, type and order of the layers stack.

Acknowledgements. This research work has been carried out in the context of the “Cátedras CONACyT” programme funded by the Mexican National Research Council.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org 1 (2015)
2. Avila-George, H., Torres-Jimenez, J., Gonzalez-Hernandez, L., Hernández, V.: Metaheuristic approach for constructing functional test-suites. IET Software 7(2), 104–117 (2013)
3. Bashiri, M., Geranmayeh, A.F.: Tuning the parameters of an artificial neural network using central composite design and genetic algorithm. Scientia Iranica 18(6), 1600–1608 (2011)
4. Bengio, Y.: Learning deep architectures for ai. Foundations and trends® in Machine Learning 2(1), 1–127 (2009)
5. Eyben, F., Wöllmer, M., Schuller, B.: Opensmile: the munich versatile and fast open-source audio feature extractor. In: Proceedings of the international conference on Multimedia. pp. 1459–1462. ACM (2010)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
7. Kacker, R.N., Kuhn, D.R., Lei, Y., Lawrence, J.F.: Combinatorial testing for software: An adaptation of design of experiments. Measurement 46(9), 3745 – 3752 (2013)
8. Kuhn, D.R., Kacker, R.N., Lei, Y.: Measuring and specifying combinatorial coverage of test input configurations. Innovations in Systems and Software Engineering pp. 1–13 (2015)
9. Leung, F.H., Lam, H.K., Ling, S.H., Tam, P.K.: Tuning of the structure and parameters of a neural network using an improved genetic algorithm. Neural Networks, IEEE Transactions on 14(1), 79–88 (2003)
10. Proust, M.: Design of Experiments Guide, Version 12. JMP, A Business Unit of SAS, SAS Campus Drive, Cary, NC27513 (2015)
11. Taguchi, G.: Introduction to quality engineering: designing quality into products and processes. ARRB Group (1986)
12. Tortum, A., Yayla, N., Çelik, C., Gökdağ, M.: The investigation of model selection criteria in artificial neural networks by the taguchi method. Physica A: Statistical Mechanics and its Applications 386(1), 446–468 (2007)

13. Tsai, J.T., Chou, J.H., Liu, T.K.: Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm. Neural Networks, IEEE Transactions on 17(1), 69–80 (2006)
14. Xiao, C., Cai, Z., Wang, Y., Liu, X.: Tuning of the structure and parameters of a neural network using a good points set evolutionary strategy. In: Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for. pp. 1749–1754. IEEE (2008)
15. Zhao, L., Qian, F.: Tuning the structure and parameters of a neural network using cooperative binary-real particle swarm optimization. Expert Systems with Applications 38(5), 4972–4977 (2011)