AUTOMATIC CLASSIFICATION OF CHAINS OF GUITAR EFFECTS THROUGH EVOLUTIONARY NEURAL ARCHITECTURE SEARCH

Michele Rossi, Giovanni Iacca, and Luca Turchet

Department of Information Engineering and Computer Science
University of Trento
Trento, IT
`michele.rossi-2@unitn.it`

ABSTRACT

Recent studies on classifying electric guitar effects have achieved high accuracy, particularly with deep learning techniques. However, these studies often rely on simplified datasets consisting mainly of single notes rather than realistic guitar recordings. Moreover, in the specific field of effect chain estimation, the literature tends to rely on large models, making them impractical for real-time or resource-constrained applications. In this work, we recorded realistic guitar performances using four different guitars and created three datasets by applying a chain of five effects with increasing complexity: (1) fixed order and parameters, (2) fixed order with randomly sampled parameters, and (3) random order and parameters. We also propose a novel Neural Architecture Search method aimed at discovering accurate yet compact convolutional neural network models to reduce power and memory consumption. We compared its performance to a basic random search strategy, showing that our custom Neural Architecture Search outperformed random search in identifying models that balance accuracy and complexity. We found that the number of convolutional and pooling layers becomes increasingly important as dataset complexity grows, while dense layers have less impact. Additionally, among the effects, tremolo was identified as the most challenging to classify.

1. INTRODUCTION

Guitar effects are crucial in shaping the sound of a guitarist, a musical piece, or an entire genre. Detecting guitar effect chains can help musicians replicate specific tones and assist producers in crafting genre-specific tracks or analyzing guitarists' unique sounds. This knowledge could also enhance Music Information Retrieval (MIR) tasks like genre recognition [1], music tagging [2], emotion recognition [3], and transcription [4]. However, guitar effect identification is less explored compared to other MIR tasks, with Deep Learning (DL)-based approaches emerging only recently [5–8].

One of the most significant contributions in this space is reported in [9], where the task of recognizing a single guitar effect is treated as a multiclass classification problem and solved using Support Vector Machines (SVM). The authors also introduced a new dataset comprising monophonic and polyphonic guitar and bass recordings processed with a single effect. A more recent approach uses a CNN to classify guitar effects, as described in [5]. In this study, the authors processed non-linear guitar effects, such as overdrive, distortion, and fuzz, using clean audio samples from [9]. A different approach was presented in [6], where two paradigms were implemented, one based on shallow neural networks, and the other based on SVM; the authors created their dataset by processing clean guitar samples from [9].

Concerning the classification of chains of guitar effects, the authors of [10] proposed a method for classifying sequences of three guitar effects. Two effects were considered for each position (a position could be empty), resulting in a total of 27 possible combinations.

One limitation of the works above lies in the simplicity of the datasets, which consist mainly of single notes rather than realistic guitar recordings. This limitation is addressed by the authors of [7], who also include a more realistic dataset (although composed of acoustic guitar recordings) in their work. Their main contribution is the proposal of a new approach for classifying chains of guitar effects using a multi-label strategy, where the presence or absence of a specific effect is encoded in binary label vectors. They compare four different CNN models for classification, limiting their analysis to a *fixed order of effects and fixed parameter values*.

In [11], the authors focused on classifying and estimating the parameters of a chain of three specific audio effects in a fixed order: distortion, tremolo, and delay. They further increased the task complexity by mixing the guitar signal with other instrument tracks. The authors of [12] proposed a method based on an autoencoder to blindly estimate the presence of three effects, focusing their analysis on an equalizer, compressor, and clipper. In [8], the authors estimate the effect chain and recover the original audio signal, focusing on four guitar effects. While their objective aligns with ours to some extent, a key distinction is that our work prioritizes the development of highly compact models, which is not a focus of their study. Similarly, outside the domain of guitar effects recognition, [13] proposed a model for blind estimation of audio processing graphs using a CNN encoder and a Transformer-based decoder. Although their model rarely achieved perfect reconstruction of the original graph, the rendered audio remained perceptually similar. While their results are notable, also in this case their approach does not emphasize compact models, making it difficult to directly compare with our work.

When it comes to using CNNs, a key challenge is designing optimal architectures, especially with limited baseline models. Neural Architecture Search (NAS) tackles this by automatically exploring hyperparameter space using metaheuristics such as Evolutionary Algorithms (EAs), inspired by evolutionary principles.

Most of the literature on EA-based NAS currently focuses on tuning CNN models for image classification, with only a few studies having explored the use of Evolutionary Algorithms (EAs) for automatically tuning hyperparameters of CNNs for audio and, more in general, time series data [14–16]. To the best of our knowledge, no research has focused thus far on applying EAs to find optimal models for electric guitar effect classification.

In this paper, we enrich the availability of guitar effect data by proposing three novel datasets of unique guitar improvisations, processed through chains of digital guitar effects plugins, that vary in order and parameters. We also introduce a method to automatically discover *highly compact* models using a custom implementation of Genetic Algorithms (GAs), comparing the results with a Random Search (RS) strategy. Finally, we compare our compact models with the one proposed in [5], the only effect classification model in the literature that shares the characteristic of reduced memory and power consumption, characterized by a small number of parameters, making it comparable to the models in our work.

2. MATERIALS AND METHODS

This Section describes three new datasets[1] of processed guitar audio tracks and presents our methodology for achieving the multi-label classification of a chain of guitar effects through an NAS approach.

2.1. Datasets

Two right-hand techniques and two different pick-up positions were used for each of the four electric guitars considered for the creation of the three datasets, namely the Paul Reed Smith Custom SE, the Epiphone Les Paul, the Fender Squier Stratocaster, and the Fender Squier Telecaster. For the pickups, we selected two positions for each guitar: the neck and the bridge. As for the right-hand techniques, we adopted two approaches: using a plectrum, also known as a pick, and finger-picking.

A professional guitarist with 20 years of musical experience was involved in the recordings. In total, 400 guitar tracks were recorded, with 100 tracks for each guitar. Each track has a fixed duration of 11.5 s and features a unique guitar improvisation that predominantly utilizes the pentatonic scale (but also diatonic scales), which is the scale that, statistically, is the most commonly used by guitarists. The four guitars were recorded considering all the possible keys, and the improvisations were performed through all the guitar neck positions.

The guitars were recorded directly via an Audient iD22 interface at a sampling rate of 44 100 Hz and 16-bit depth. Tracks were recorded using Reaper, normalized in loudness, and exported. File names encoded details like guitar type, pickup position, and technique (e.g., `tele_bridge_pick19.wav`).

Each guitar track was processed with up to five effects in cascade, namely: overdrive, chorus, tremolo, delay, and reverb. For each unprocessed guitar track, all the possible combinations of these five effects were considered. In particular, we created three different datasets with increasing complexity in terms of order and configuration of the effects, which are described in detail in the following.

DS #1 The first dataset considers a fixed sequence of guitar effects. In this case, the order was chosen according to the general rules that a guitarist usually considers when defining the configuration of his/her guitar effects. Obviously, there is no absolute rule for their order of applications and for their configuration, but we may say that there are some conventions to be followed when dealing with a cascade of effects. A plausible sequence was determined to be: overdrive, chorus, tremolo, delay, and reverb. Another constraint of DS #1 concerns the effects' parameters. For this dataset, a predefined value for each parameter was chosen according to the default value of each effect (see Table 1). All the plugins we used to implement guitar effects for dataset creation are freely available. For the chorus, delay, and reverb effects, we used the implementation provided by the `pedalboard` library by Spotify. The overdrive was The Klone By Fazertone, a simulation of the Klon Centaur overdrive, and the tremolo was the Mtremolo by MeldaProduction.

[1]Available at:https://doi.org/10.5281/zenodo.7871720

Effect	Parameter	DS #1	DS #2	DS #3
Overdrive	Gain	0.5	0.2 - 1.0	0.2 - 1.0
Chorus	Mix	0.5	0.2 - 0.5	0.2 - 0.5
Tremolo	Rate	4	2 - 10	2 - 10
Delay	Time	0.5	0.3 - 0.6	0.3 - 0.6
Reverb	Room	0.5	0.2 - 0.7	0.2 - 0.7

Table 1: Effects' parameters with their corresponding range. For every effect, the range is in $[0, 1]$, except for the tremolo whose possible values range between 0.1 and 20 and are expressed in Hz.

DS #2 The second dataset overcomes the limitation of DS #1 by considering variability in the most relevant parameter of each effect. As said, guitar effects typically have multiple knobs in their circuit, each having a different impact in terms of relevance on the final sound. For example, most of the time, the overdrive presents three parameters, but one of those (the gain) is by far more relevant than the other two (tone and volume) in characterizing its specific timbre. For this dataset, the specific value of the main parameter of each effect was randomly sampled as a continuous value from a uniform distribution in a predefined range, for a total of 6400 values for each effect. These ranges were carefully tuned based on informal perceptual tests (see Table 1). This dataset makes the classification task more complex, keeping the constraint of having a predefined sequence of guitar effects in the chain.

DS #3 The limitation of DS #2 is addressed in the third dataset, which also considers a variable order of effects in the chain. From an implementation perspective, the first step in constructing this dataset involved including all possible combinations of the five effects for each audio file, as was done for DS #2. Then, the effects' positions were shuffled before processing each guitar track. This procedure expands significantly the possible chain configurations, also exploiting sequences of effects that are less relevant from a guitarist's point of view. Nevertheless, this new dataset can force a better generalization in the CNN, which must learn to recognize the presence of specific effects despite their particular order.

For each dataset, the total number of audio files after the application of the effects' chains is 12 800, i.e., 400 audio files processed with 2^5 effect combinations. Considering that during the preprocessing phase, detailed in Section 2.2, each audio file is segmented into five parts, the final number of training samples is 64 000, corresponding to more than 35 hours of processed guitar tracks for each dataset. The presence of a particular effect is encoded in binary format in the name of each audio file.

2.2. Preprocessing

Initially, each guitar recording was down-sampled from 44 100 to 22 050 Hz, and, as previously mentioned, divided into five segments of 2 s each, leading to a total of 64000 audio files. Subsequently, each segment was transformed into its corresponding mel-spectrogram, consisting of a 2-dimensional representation of the audio signal, representing both the time and the frequency information with a single matrix. The window size and the hop length were set to 2048 and 512 samples, respectively, while the number of mels, which is related to the frequency information, was set to 128.

2.3. Convolutional Neural Network

The model used for the multi-label classification task is a CNN with a variable structure that depends on the specific hyperparameters considered, which are discovered by using the proposed NAS approach. At a high level, the model is divided into two main parts: a set of convolutional blocks and a set of dense blocks. The former part of the model is composed of at least one convolutional layer and one max-pooling layer stacked together, which are always present in the architecture, and of up to four subsequent convolutional and pooling layers, again, stacked together, whose presence, instead, depends on the specific hyperparameters. The latter part of the model is made up of one or more (up to three) dense blocks (i.e., fully connected layers), followed by the output layer. The latter contains five neurons, which correspond to the five possible effects of the multi-label classification task.

In addition, batch normalization is implemented for each convolutional layer (placed after the max-pooling layer), and dropout regularization is applied on each dense layer except for the output. The Rectified Linear Unit is the activation function of choice for the network because of its robustness for gradient vanishing and its fast convergence. The activation function for the output layer is the sigmoid, which outputs a real value in the range $(0, 1)$, representing the probability associated with the presence of a specific effect in the audio file.

The other main hyperparameters of the CNN, i.e., the number of neurons of the dense layers, the number of filters (i.e., output channels) of the convolutional layers, and the dropout probabilities, are discovered during the NAS optimization phase, described in detail in Section 2.4. The number of weights in the CNN varies based on the specific configuration of hyperparameters found by the NAS, ranging from a minimum of 167 up to a maximum of approximately 46.5 million.

2.4. Proposed GA-based NAS

Figure 1 illustrates the genotypes managed during the NAS process, while Table 2 lists the hyperparameters considered during the encoding phase along with their ranges. Key hyperparameters include the number of dense and convolutional blocks, which influence the CNN's depth and task-specific tuning; the number of filters, which affect feature extraction capabilities; the x and y kernel dimensions, optimized independently for time and frequency representation in spectrograms; and the dropout probability in dense layers, which aids in reducing overfitting.

Concerning the functions and operators, the fitness function evaluates the performance of each candidate CNN model, which is encoded by each individual generated during the evolutionary process, based on the accuracy achieved on the validation set after training that model on the training set. Further details are provided below. We also developed a custom initialization mechanism that randomly generates the first population of individuals with a bias towards smaller networks. This behavior is achieved by imposing a restricted range for some hyperparameters for the first population. To be specific, the number of convolutional filters and the number of neurons of the dense layers are sampled in the restricted interval

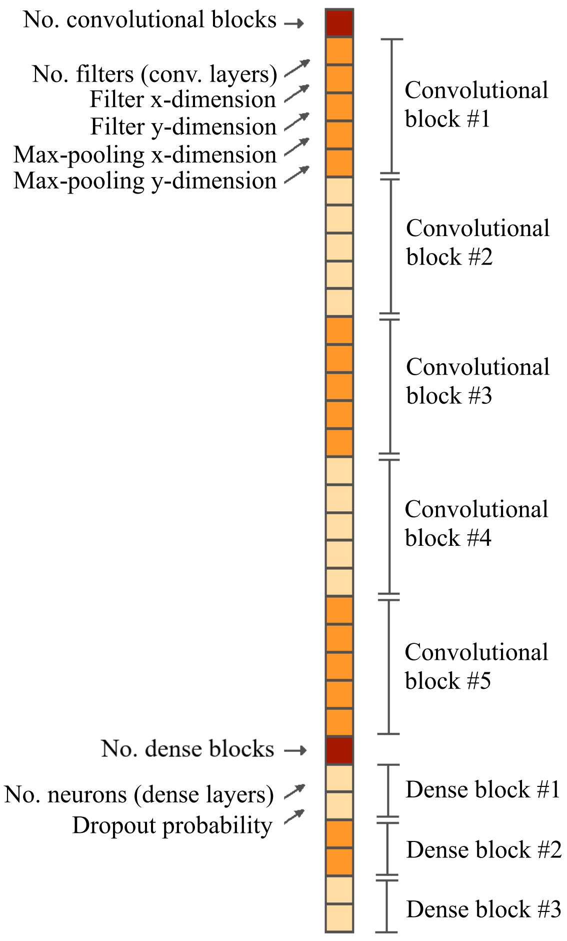

Figure 1: Graphical representation of the fixed-length individual genotype describing a CNN model for the multiclass classification task. The genotype contains 3 continuous variables, which represent the dropout probabilities of up to 3 dense blocks, and 30 integer variables: 5×5 parameters for the convolutional blocks (i.e., 5 parameters for up to 5 blocks), 1 for the no. of convolutional blocks that are actually used (at least one), 1×3 parameters for the dense blocks (i.e., 1 parameter, the no. of neurons, for up to 3 blocks), 1 for the no. of dense blocks that are actually used (at least one). The parameters for the blocks that are not used are ignored when it comes to translating a genotype into an actual CNN model.

Table 2: Hyperparameters with their corresponding range.

Hyper-parameter	**Minimum**	**Maximum**
No. convolutional blocks	1	5
No. filters (conv. layers)	2	64
Filter x-dimension	2	7
Filter y-dimension	2	7
Max-pooling x-dimension	1	2
Max-pooling y-dimension	1	2
No. dense blocks	1	3
No. neurons (dense layers)	2	64
Dropout probability	0.0	0.5

$[2, 8]$ for the initial population. See Table 2 for the normal range of each hyperparameter. This mechanism aims at finding CNNs' architectures with a low number of parameters, i.e., weights, for the purpose of lessening the computational consumption and the inference time. Selection of parent solutions is performed using a roulette-wheel principle [17], with higher accuracy individuals having a greater chance of selection. Moreover, the elitism mechanism, i.e., the process of preserving the best-performing individual from one generation to the next, is added. We implemented the binary crossover to facilitate information exchange between two selected genotypes.

The mutation operator modifies one or more values in an individual's vector with a certain probability to introduce diversity and explore the solution space. In our NAS, this is adapted for genes with limited value ranges, such as block counts and filter dimensions, where mutation involves sampling a new value from the range. For hyperparameters like the number of filters, neurons, and dropout probability, mutation adds an integer or continuous value to the gene.

3. EXPERIMENTAL SETUP

Our evaluation setup comprises two phases: in the first one, we perform the NAS process to find the best-performing models. In the second phase, we analyze the performance of these models in greater detail. We also provide a comparison with the model implemented in [5], to prove the effectiveness of our NAS strategy. These two phases have been conducted as follows.

3.1. Neural Architecture Search

For each of the three datasets, the GA-based NAS was performed considering a number of generations of 16 and a population size of 8. The crossover probability was set to 0.8, while the mutation probability was set to 0.3. These parameters have been set empirically after a set of preliminary experiments. For each dataset, 4 runs of the GA were performed to collect statistics on the behavior of the algorithm.

During the NAS process, each candidate architecture was trained on a training set and then its accuracy was evaluated on a validation set to find the corresponding fitness. More specifically, three guitars, *les*, *prs*, and *tele*) were used for training, and the remaining one (*strat*) for validation. This train/validation splitting approach has been chosen with the aim of ensuring that our model has the capability to generalize well, even for guitars that it has never encountered before. In this phase, the validation accuracy was determined based on a prediction being classified as correct only when the *entire chain* is correctly detected.

A Random Search (RS) approach was implemented to provide a baseline for the search capabilities of our GA. The RS generates each solution by performing a direct sampling for each gene, i.e., hyperparameter, from the available range (see Table 2 for details). For a fair comparison with the GA, for each run of the RS, we considered the best individual found at each batch of 8 solutions, for a total of 16 batches corresponding to 128 individuals. Thus, the total budget of the RS is the same as the one used for the GA.

3.2. Classification

In the second phase, for each dataset, the best model discovered by the GA has been considered for computing all the statistics associated with the multi-label classification task. The chosen model was not simply the one that achieved the best performance during the NAS phase, but we recomputed the accuracy values associated with the best 2 models found by the GA to determine the final model for each dataset (see Table 3). This approach has been chosen for two main reasons: firstly, as previously elaborated, during the NAS, the accuracy was calculated considering only one validation set, the *strat* guitar, and only a single run. Conversely, in this second stage, we averaged 12 runs considering 4 different validation sets, leading to more accurate results for properly choosing the candidate model (the details of the cross-validation technique adopted will be explained later in this Section). Secondly, the accuracy values found during the GA can be very similar for different solutions, as it can be noticed considering, for example, the individuals of the first dataset at indices 6 and 7 of Table 3. In this case, the difference in accuracy is insignificant, about 0.1%, and recomputing the accuracy with a more precise approach may lead to a more reliable choice.

The models selected for the classification phase can be found in Table 3 at indices 7, 10, and 10 for DS #1, DS #2, and DS #3, respectively. As can be noticed, for the second dataset, the selected individual was not the one that achieved the highest accuracy value during the NAS phase.

As previously mentioned, the CNN obtained using the hyperparameters encoded in the selected individual for each dataset was trained and validated considering a specific implementation of cross-validation, particularly suited for verifying the generalization of the model. More specifically, a guitar cross-validation was implemented, where the samples of three guitars were involved in the training phase, while the samples of the remaining guitar were used for the validation phase. Therefore, in this case, four different folds, i.e., validation sets, were considered, differently from what happened during the NAS phase, where only one guitar, the *strat*, was used for the validation. For achieving more meaningful results, for each validation set, i.e., for each of the four guitars, 3 runs were performed, thus for a total of 12 runs.

The prediction values of each audio segment were fused according to the majority voting technique for computing the final model's prediction. In fact, each sample represents a 2 s segment of a specific guitar track, which was divided into five segments during the preprocessing phase. Majority voting was applied separately for each effect, considering an effect present in the audio file if at least 3 out of 5 segments detected it. Notably, this technique was excluded from the NAS phase to save computational resources and time.

For the training setup, the loss function was the binary crossentropy, optimized with Adam. The learning rate and the number of epochs were set to 5×10^{-5} and 15, respectively, and the batch size was set to 64. All these values were set according to some preliminary experiments.

As previously mentioned, we conducted a final comparison between the best models identified for each dataset and the CNN proposed in [5], which represents the most recent state-of-the-art deep learning model specifically designed for guitar effects classification while meeting the requirement of compactness (i.e., a reduced number of parameters), which is the primary objective of our work.

4. EXPERIMENTAL RESULTS

We present the NAS results to provide insight into the proposed GA's effectiveness and to assess the performance of the best-discovered model. We developed our algorithm in `Python`, using the `Keras` API with `Tensorflow` backend for both the NAS and the classification phase[2].

4.1. Neural Architecture Search

Both the NAS and the RS, used as a reference, required between 8 and 16 hours on a Tesla T4 GPU to complete a full run, which involved training and validating 128 architectures, organized into 16 batches of 8 individuals each.

Fig. 2 shows the minimum, median, and maximum values of the best solutions found at each generation across the 4 available runs of the GA for the three datasets at hand. It is important to recall that the accuracy value internally considered during the NAS process is based on the detection of the *entire chain* of effects, such that a sample is considered correct only if all the effects in the chain are detected accurately.

Concerning the performance on each specific dataset, Fig. 2 (left) depicts the performance of the GA on DS #1. Note that the algorithm discovers very promising results just after the first $4-5$ generations, suggesting that small architectures are capable of effectively handling the somewhat limited complexity of DS #1.

As depicted in Fig. 2 (center and right), the algorithm required a greater number of generations to attain satisfactory accuracy results for DS #2 and DS #3. These two datasets are incrementally more complex than DS #1, resulting in the algorithm struggling even more to discover optimal solutions during the initial generations. Yet, the NAS process still allows the discovery of well-performing architectures in later generations.

The GA was then compared to the RS approach for the optimization of the hyperparameters. The accuracy and the size, specifically the number of weights, of each CNN found by the GA and the RS were taken into consideration for the comparison.

The comparison between the architectures found by the two search strategies is depicted in Fig. 3 for the three datasets. In each scatter plot, the x and y dimensions are associated with the accuracy and the number of parameters of each solution, respectively. In the figure, the larger dots indicate the non-dominated solutions found in all runs of both RS and GA, i.e., the Pareto front that encompasses the best solutions in terms of number of parameters and accuracy found by both search strategies considered together. The numerical values associated with these non-dominated solutions are reported in Table 3.

Table 3: Non-dominated solutions found by GA and RS on the three datasets.

Dataset	Index	Optimizer	Accuracy	Parameters
DS #1	1	GA	0.245	8091
	2	GA	0.908	20 762
	3	GA	0.929	29 811
	4	GA	0.941	32 099
	5	GA	0.945	68 339
	6	GA	0.950	93 464
	7	GA	0.951	209 498
	8	RS	0.955	219 964
	9	RS	0.964	517 160
DS #2	1	GA	0.179	10 346
	2	GA	0.219	11 732
	3	GA	0.417	20 906
	4	GA	0.428	27 533
	5	GA	0.470	28 712
	6	GA	0.534	56 426
	7	GA	0.593	75 993
	8	RS	0.801	82 780
	9	GA	0.812	194 717
	10	GA	0.834	227 240
	11	GA	0.857	275 372
	12	RS	0.868	410 836
DS #3	1	GA	0.174	4778
	2	GA	0.178	6063
	3	GA	0.216	6119
	4	GA	0.409	13 651
	5	GA	0.414	34 714
	6	GA	0.419	36 671
	7	GA	0.461	45 406
	8	GA	0.637	46 162
	9	GA	0.764	50 816
	10	GA	0.791	68 213
	11	RS	0.815	224 833

As previously explained, our implementation of the GA is capable of discovering highly promising solutions with a minimal set of parameters (please note the logarithmic scale on the y-axis), especially when working with DS #1. By analyzing Fig. 3 (top) and Table 3, it can be observed in fact that out of the nine solutions on the final Pareto front, the GA was responsible for discovering seven. Nevertheless, the RS found two solutions that can be found at indices 8 and 9 of Table 3 with a slight increase in accuracy.

When considering DS #2, we can notice how the GA exhibits a variety of solutions belonging to the overall Pareto front, see the larger dots in Fig. 3 (center). In this case, the RS was able to identify the solution with the highest accuracy value, with an increase of approximately 1% compared to the best individual discovered by the GA, but at the cost of increasing the number of parameters, i.e., weights, of the model by about 50%. Moreover, as can be noticed, the non-dominated solutions found by the GA are spread across the scatter plot, including many solutions with a relatively low accuracy value. This outcome can be attributed to the fact that the models discovered during the first generations by the GA have a very limited number of convolutional output channels (i.e., filters) and neurons, resulting in highly compact architectures that lack the capacity to effectively handle the complexity of DS #2.

This aspect is even more evident when looking at the results

[2]The code is available at `https://github.com/michelerossi1/Paper_effects_chain`.

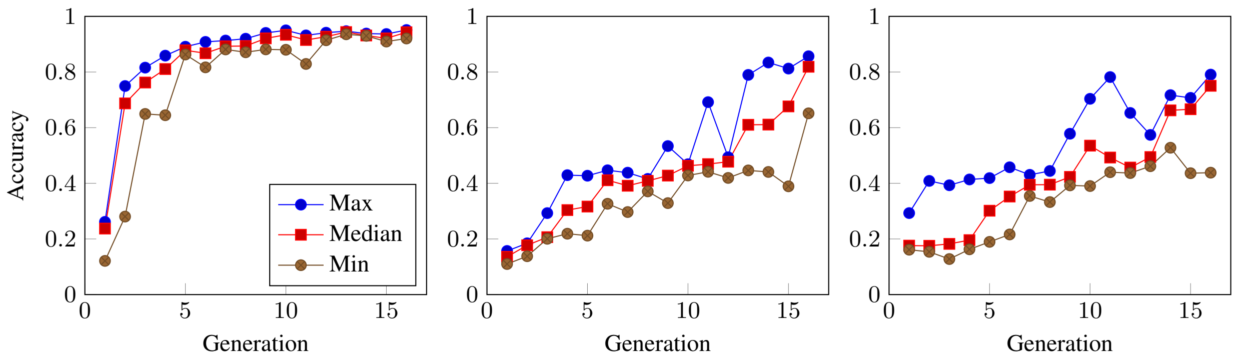

Figure 2: Minimum, median, and maximum fitness values for each generation on DS #1 (left), #2 (center), and #3 (right).

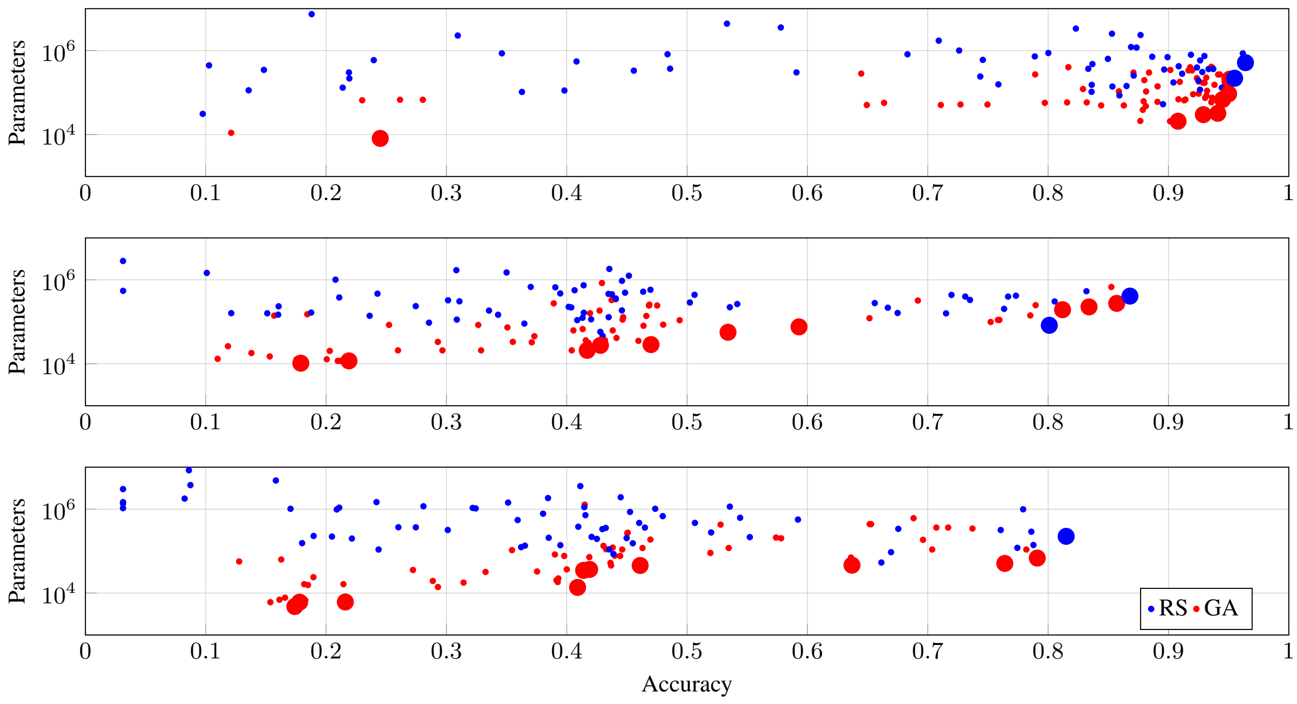

Figure 3: Distribution of the architectures found in the various runs of the GA and the RS on DS #1 (up), DS #2 (center), and DS #3 (bottom). The larger dots correspond to the Pareto front architectures listed in Table 3.

shown in Table 3, where all the non-dominated solutions found by the two search strategies are reported. In contrast to the outcome found on DS #1, we can observe that the solutions found by the GA cover an area of the search space associated with relatively low accuracy values, differently from the RS, which instead finds non-dominated solutions with accuracy values higher than 0.8.

Finally, the results associated with DS #3 can be analyzed considering Fig. 3 (bottom). It can be observed that, similarly to the other two datasets, the solutions found by the GA tend to use fewer parameters than those found by the RS. Once again, the majority of the Pareto front is composed of solutions discovered by the GA, indicating that the GA successfully managed both the accuracy and the model size throughout the search process. However, the RS approach discovered a solution that resulted in a slight increase in accuracy compared to the best model found by the GA, but at the cost of tripling the number of parameters in the model.

4.2. Classification

For each dataset, the top-performing model identified during the GA-based NAS process was selected following the approach detailed in Section 3, and the F1-scores for each effect were calculated (see Fig. 4 and Table 4). The plots illustrate the challenges faced by the CNN in detecting each specific effect. These scores were derived by averaging the results from 12 runs of the classification task, each involving a full training and validation process using the previously described guitar-cross-validation technique. As previously stated, Table 4 presents the comparison between the

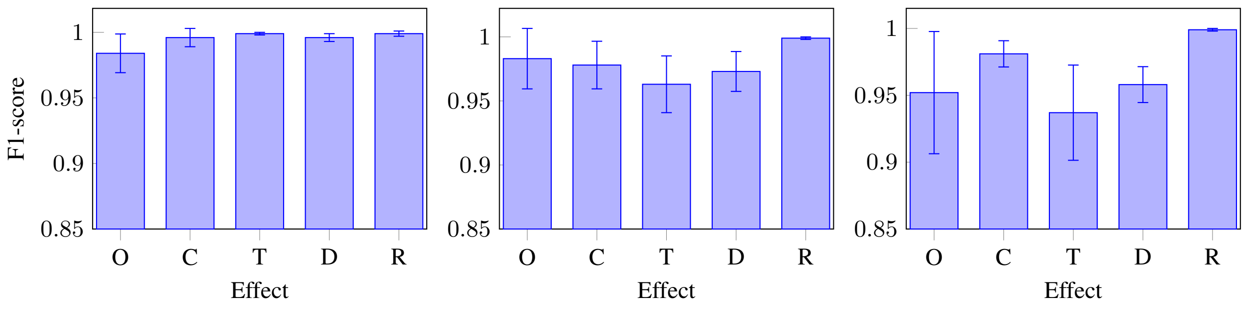

Figure 4: F1-score value for each effect on DS #1 (left), DS #2 (center), and DS #3 (right). O = overdrive, C = chorus, T = tremolo, D = delay, R = reverb.

Table 4: Mean and standard deviation of the F1-score for each effect and dataset.

Dataset	Model	Overdrive	Chorus	Tremolo	Delay	Reverb
DS #1	Our CNN	0.984 ± 0.015	0.996 ± 0.007	0.999 ± 0.001	0.996 ± 0.003	0.999± 0.002
	Comunità et al. [5]	0.927 ± 0.106	0.982 ± 0.032	0.995 ± 0.003	0.930 ± 0.039	0.994± 0.010
DS #2	Our CNN	0.983 ± 0.024	0.978 ± 0.019	0.963 ± 0.023	0.973 ± 0.016	0.999 ± 0.001
	Comunità et al. [5]	0.889 ± 0.098	0.962 ± 0.022	0.704 ± 0.094	0.855 ± 0.054	0.990 ± 0.016
DS #3	Our CNN	0.952 ± 0.048	0.981 ± 0.010	0.937 ± 0.038	0.958 ± 0.014	0.999 ± 0.001
	Comunità et al. [5]	0.914 ± 0.070	0.966 ± 0.014	0.759 ± 0.044	0.827 ± 0.070	0.986 ± 0.025

top models identified by our NAS and the model [5], hereafter referred to as the *baseline* model.

We conducted a Wilcoxon signed-rank test that demonstrated that our models generally outperformed the baseline model across multiple effects and datasets. In DS #1, our model showed significant improvements for only two effects: tremolo and delay. In DS #2, our model surpassed the baseline in all five effects. Lastly, in DS #3, our model significantly outperformed the baseline in all effects except for Overdrive.

When considering these results, it is important to note that the baseline model, despite being very similar to our models, was originally designed for a somewhat different task: multi-class classification of non-linear effects. Thus, this comparison should be interpreted within that context. Nevertheless, it offers insights into the potential for converting the model from single-class to multi-class while maintaining high accuracy.

5. DISCUSSION

The proposed GA showed promising results across the three datasets. However, it found good solutions within a limited number of generations only for DS #1, requiring more generations for DS #2 and DS #3. This is likely due to the algorithm starting with small architectures in early generations, consuming resources on models insufficiently capable of handling the complexity of DS #2 and DS #3.

Additionally, we discovered that certain hyperparameters played a crucial role in determining the optimal CNN models. In particular, considering DS #2 and DS #3, we identified trends in hyperparameter configurations leading to high accuracy values. Analyzing models with accuracy greater than 0.7, we observed significant similarities in three parameters: the number of convolutional layers, the number of dense layers, and the size of max pooling filters. All 15 architectures that achieved accuracy exceeding 0.7 (9 for DS #2 and 6 for DS #3) in the 4 runs of the NAS had 5 convolutional layers. Additionally, none of them had 3 dense layers, and the choice between having 1 or 2 dense layers showed no clear advantage. Regarding the size of max pooling layers, the (1×1) case, which represents no pooling layer, was rare, occurring only in 6 out of 75 potential max pooling layers (i.e., 15 architectures with 5 layers each). In contrast, the models demonstrated very high performance across various combinations of hyperparameters for DS #1, except for the number of convolutional layers. Out of the 56 architectures that achieved an accuracy greater than 0.7, 53 had either 4 or 5 convolutional layers, with no evident preference between these two values.

Other observations can be made in relation to each specific guitar effect. Considering DS #1, the best model found was able to perform the correct detection of each effect without difficulty. The overdrive was the only effect for which the CNN exhibited a lower performance, with an F1-score of 0.98. For DS #2, the accuracy of the model reached its lowest value for the classification of the tremolo effect, obtaining an F1-score of 0.96. This behavior can be interpreted considering that this specific effect, which is basically an amplitude modulation, can vary considerably in its oscillation rate, considering different samples. In DS #2, as previously explained, the model can encounter during the validation phase samples with different tremolo rates with respect to the ones encountered during training.

As can be noticed in Fig. 4 (right), the model decreased its performance for almost every effect when considering DS #3. This behavior can be understood considering that the specific order of the effects in the chain significantly impacts the final sound. Therefore, the CNN should be able to recognize the presence of a specific effect independently of its internal settings, of the presence of other effects, and of the specific position the effect occupies in the chain, leading to a more challenging task.

This study has several limitations that could be addressed with future work. The datasets rely on specific guitar plugins for each effect, which may limit the model's accuracy if different plugins are used. This could be mitigated by creating a larger, more diverse dataset with plugins from various manufacturers. Another limitation involves parameter selection, as effects that have more than one influential parameter require a broader sampling of their settings. To address this, an enhancement to the current work would be to randomly sample every parameter for each implementation of each effect involved in the dataset. Additionally, the analysis focused on five common effects (overdrive, chorus, tremolo, delay, and reverb), but could be expanded to include others like compression, distortion, and vibrato, as well as longer effect chains. Finally, incorporating performances from more guitarists could improve the model's generalization.

6. CONCLUSIONS

This work addresses limitations in guitar effect recognition by proposing three novel datasets of realistic audio tracks and implementing a custom Neural Architecture Search (NAS) approach using Genetic Algorithms (GA) to optimize Convolutional Neural Network (CNN) hyperparameters. We recorded 400 guitar improvisations played on four different guitars, processed through chains of five effects with increasing complexity: fixed settings (DS #1), parameter variations (DS #2), and both order and parameter variations (DS #3).

Through our NAS approach, we successfully identified CNN architectures that achieve a balance between accuracy and efficiency, making them more suitable for real-time or resource-constrained applications. Additionally, we found that as dataset complexity increases, the depth of convolutional and pooling layers plays a crucial role in maintaining classification performance, while dense layers have a lesser impact. Among the effects studied, the tremolo proved to be the most difficult to classify.

Overall, this study provides a foundation for developing more practical and efficient models for effect chain estimation, paving the way for enhanced real-time applications in music technology.

7. REFERENCES

[1] Yang Yu, Sen Luo, Shenglan Liu, Hong Qiao, Yang Liu, and Lin Feng, "Deep Attention Based Music Genre Classification," *Neurocomputing*, vol. 372, pp. 84–91, 2020.

[2] Jordi Pons and Xavier Serra, "musicnn: Pre-Trained Convolutional Neural Networks for Music Audio Tagging," in *Late-breaking/demo session in International Society for Music Information Retrieval Conference*, 2019.

[3] Luca Turchet and Johan Pauwels, "Music Emotion Recognition: Intention of Composers-Performers versus Perception of Musicians, Non-Musicians, and Listening Machines," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 305–316, 2021.

[4] Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert, "Automatic Music Transcription: An Overview," *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 20–30, 2018.

[5] Marco Comunità, Dan Stowell, and Joshua D Reiss, "Guitar Effects Recognition and Parameter Estimation with Convolutional Neural Networks," *Journal of the Audio Engineering Society*, vol. 69, no. 7/8, pp. 594–604, 2020.

[6] Henrik Jürgens, Reemt Hinrichs, and Jörn Ostermann, "Recognizing Guitar Effects and Their Parameter Settings," in *International Conference on Digital Audio Effects*, 2020.

[7] Jinyue Guo and Brian McFee, "Automatic Recognition of Cascaded Guitar Effects," in *International Conference on Digital Audio Effects*, 2023.

[8] Osamu Take, Kento Watanabe, Takayuki Nakatsuka, Tian Cheng, Tomoyasu Nakano, Masataka Goto, Shinnosuke Takamichi, and Hiroshi Saruwatari, "Audio Effect Chain Estimation and Dry Signal Recovery from Multi-Effect-Processed Musical Signals," in *International Conference on Digital Audio Effects*, 2024, pp. 1–8.

[9] Michael Stein, Jakob Abeßer, Christian Dittmar, and Gerald Schuller, "Automatic Detection of Audio Effects in Guitar and Bass Recordings," in *Audio Engineering Society Convention*. 2010, Audio Engineering Society.

[10] Michael Stein, "Automatic Detection of Multiple, Cascaded Audio Effects in Guitar Recordings," in *International Conference on Digital Audio Effects*, 2010.

[11] Reemt Hinrichs, Kevin Gerkens, Alexander Lange, and Jörn Ostermann, "Convolutional Neural Networks for the Classification of Guitar Effects and Extraction of the Parameter Settings of Single and Multi-Guitar Effects from Instrument Mixes," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2022, no. 1, pp. 28, 2022.

[12] Côme Peladeau and Geoffroy Peeters, "Blind Estimation of Audio Effects using an Auto-Encoder Approach and Differentiable Digital Signal Processing," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2024, pp. 856–860, IEEE.

[13] Sungho Lee, Jaehyun Park, Seungryeol Paik, and Kyogu Lee, "Blind Estimation of Audio Processing Graph," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2023, pp. 1–5, IEEE.

[14] Domenico Stefani and Luca Turchet, "Bio-Inspired Optimization of Parametric Onset Detectors," in *International Conference on Digital Audio Effects*. 2021, pp. 268–275, IEEE.

[15] Özkan İnik, "CNN Hyper-Parameter Optimization for Environmental Sound Classification," *Applied Acoustics*, vol. 202, pp. 109168, 2023.

[16] Ali A Samir, Abdullah R Rashwan, Karam M Sallam, Ripon K Chakrabortty, Michael J Ryan, and Amr A Abohany, "Evolutionary Algorithm-Based Convolutional Neural Network for Predicting Heart Diseases," *Computers & Industrial Engineering*, vol. 161, pp. 107651, 2021.

[17] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra, "Comparative Review of Selection Techniques in Genetic Algorithm," in *International Conference on Futuristic Trends on Computational Analysis and Knowledge Management*. 2015, pp. 515–519, IEEE.