AUTOMATIC RECOGNITION OF CASCADED GUITAR EFFECTS

Jinyue Guo *

RITMO, Department of Musicology
University of Oslo
Oslo, Norway
jinyue.guo@imv.uio.no

Brian McFee

Music and Audio Research Laboratory, MARL
New York University
New York, USA
brian.mcfee@nyu.edu

ABSTRACT

This paper reports on a new multi-label classification task for guitar effect recognition that is closer to the actual use case of guitar effect pedals. To generate the dataset, we used multiple clean guitar audio datasets and applied various combinations of 13 commonly used guitar effects. We compared four neural network structures: a simple Multi-Layer Perceptron as a baseline, ResNet models, a CRNN model, and a sample-level CNN model. The ResNet models achieved the best performance in terms of accuracy and robustness under various setups (with or without clean audio, seen or unseen dataset), with a micro F_1 of 0.876 and Macro F_1 of 0.906 in the hardest setup. An ablation study on the ResNet models further indicates the necessary model complexity for the task.

1. INTRODUCTION

The task of guitar effect recognition is to build an algorithm that recognizes which kinds of effects are used in a given piece of guitar audio. It is common to see multiple, nonlinear effects cascaded to produce a rich guitar timbre. This makes it difficult to build an effective recognition algorithm. Additionally, unlike some other music information retrieval (MIR) tasks such as pitch tracking or music tagging, resources for guitar effect recognition are relatively limited. The lack of data and evaluation standards makes it difficult to standardize the setup.

As we review in section 2, previous works on guitar effect recognition have framed it as a typical classfication problem. However, most previous research tend to form the question as a multi-class but single-label classification task, which means the models either work on samples with single effects or consider a group of effects as one label. However, in practice, different types of linear or nonlinear effects are often cascaded to create the final output. A second challenge is that the audio samples only contain a single sound event, either a single pluck or a single sweep, which makes the performance unpredictable on common guitar recordings that usually contain more complex temporal and spectral components.

1.1. Our Contributions

In this paper, we constructed the task of guitar effect recognition as a multi-label classification task by applying two improvements: First, we created a workflow that renders arbitrary guitar recordings with all combinations of effects using `SoX`. Second, we adapted several Neural Network models that can handle the multi-label classification task. The models were modified and benchmarked with several task setups: using the original clean audio as a hint or not, and zero-shot classification on unseen data distributions. As a complement of this paper, source-code and result sheets are provided online[1][2].

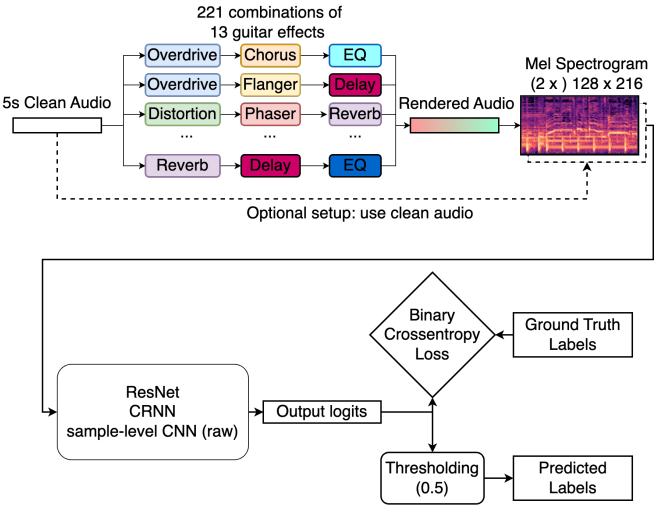


Figure 1: *Proposed method of effect rendering and multi-label classification.*

2. RELATED WORKS

The question of guitar effect recognition was first formed into a classification task by Stein et al. [1, 2]. They formed a single-label classification task by using only one effect at a time [1] or only classifying the group of effects [2]. The classifiers are built with hand-crafted audio features such as spectral centroid and cepstral features, and then feed into a Support Vector machine for classification. They also introduced a new dataset, *IDMT-SMT-Audio-Effects*. The dataset was manually processed using a Digital Audio Workstation, containing 55,000 samples of processed guitar audio files, recorded with two electric guitars and two electric basses. The samples are monophonic or polyphonic, but each includes a single sound event only.

Schmitt and Schuller [3] continued on the direction of hand-crafted audio features with a comprehensive research on features,

* This paper is based on the master's thesis research performed by the first author while studying at New York University

[1] https://github.com/fisheggg/SFXlearner
[2] DOI: 10.5281/zenodo.7973536

Table 1: *Effects used, the corresponding* `pysox` *function, and parameters. Parameters not listed are default values in* `pysox`.

Effect type	Effect name	pysox function	parameters
Non-linear	Overdrive	*sox.overdrive()*	gain_db: 5
	Distortion	*sox.overdrive()*	gain_db: 15
Modulation	Chorus	*sox.chorus()*	n_voices: 5
	Flanger	*sox.flanger()*	depth:5, phase:50
	Phaser	*sox.phaser()*	default
	Tremolo	*sox.tremolo()*	default
Ambience	Reverb	*sox.reverb()*	reverberance: 80
	Feedback delay	*sox.echos()*	n_echos: 3 delays: [200,400,600] decays:[0.4,0.2,0.1] gain_out:0.5
	Slapback delay	*sox.echo()*	n_echos: 3 delays: [200,400,600] decays:[0.4,0.2,0.1] gain_out:0.5
EQs	Low boost	*sox.bass()*	frequency: 200 gain_db: 10
	Low reduct	*sox.bass()*	frequency: 200 gain_db: -10
	High boost	*sox.treble()*	frequency: 8000 gain_db: 20
	High reduct	*sox.treble()*	frequency: 8000 gain_db: -20

including *zero-crossing rate* (ZCR), *root-mean-square* (RMS) energy, etc. They also introduced the concept of *Bag-of-Audio-Words* (BoAW), which generates a codebook of frame-level features. The classification setup and dataset are the same as Stein et al. [1, 2].

Neural network methods were first introduced to the task by Jürgens et al. [4] and Comunità, M. et al. [5]. Both of these works attempted to recognize not only the types of effects but also the parameters. Since the *IDMT-SMT-Audio-Effects* dataset did not provide any parameter information, both of these works have created their own setups. Jürgens et al. [4] continued to use the aforementioned effect settings [1] and built specific *Multi-Layer Perceptron* (MLP) [6] parameter regressor for each class after being classified. On the other hand, Comunità et al. [5] used a different setting. Instead of using the 10 classes [1], they focused on nonlinear effects. Thirteen overdrive, distortion and fuzz effect units are selected and tested on unified parameters (Gain and Tone/EQ). Instead of building specific regressors for each effect, a uniform *Convolutional Neural Network* (CNN) is trained to classify the effects and estimate their parameters.

Meanwhile, some of the latest neural network methods have been introduced to similar Music Information Retrieval tasks. The ResNet model [7] was originally proposed for Computer Vision tasks, but is proven to be also effective on genre recognition [8]. The CRNN model [9] combines the idea of convolutional neural network and recurrent neural network, achieved great results on music classification. Unlike the previous models that use spectrogram as their input, the Sample-Level CNN model [10, 11] applied 1-D convolution on raw waveforms and achieved great results on music auto-tagging. These models have been proven on solving audio-based multi-label classification problems, but are not adapted to guitar effect recognition yet.

3. DATA PREPARATION

3.1. Clean Datasets

Instead of recording sounds from physical guitar effect units, we chose to use clean guitar audio datasets, and manually render audio effects. There are two main benefits of this process. The first is to have huge flexibility in controlling the types, numbers, orders, and parameters of the effects we apply to each audio. The richness of data variation can contribute to the robustness of the model. The second benefit is that we can get a clean version of the processed audio, which can be used as a reference in the model.

We use two different clean guitar datasets in our rendering progress. `GuitarSet` [12] contains more than 10,000 seconds of guitar audio recordings, played by 6 different players in 5 different styles. The variety of this dataset can make our model robust on the timbre difference of the guitars and play styles of different guitarists. However, the instrument and recording setting of `GuitarSet` is fixed. As a complement, the fourth subset of `IDMT-SMT-Guitar` dataset [13] contains 384 guitar samples of 64 different music pieces that are played in 2 different tempi and 3 different guitar models, and recorded with two different setups. The audio files are sliced into 5-second excerpts with the tails dropped. After the process, we have 2004 samples from `GuitarSet`, and 650 samples from `IDMT-SMT-Guitar`.

3.2. Effect Selection and Dataset Rendering

We use `SoX` [14] and its python wrapper `pysox` [15] to create the pipeline and render effects to the clean datasets. We chose 13 often-used sound effects, including nonlinear effects, modulation, and EQs. Compared to the standard effect types proposed by Stein et al. [1], we added EQ effects since they are often used

in pedalboards. The *vibrato* effect is replaced by *chorus*, since it could be considered as a special case of *chorus* when the number of voices equals to 1. One limitation here is that `SoX` only provides one clipping algorithm, hence the *Overdrive* and *Distortion* actually use the same function but with different values of gain. The parameters of effects are hand-picked to provide audible changes to the audio samples and are fixed during the generation process. Table 1 shows the selected effects.

Using the `pysox` API, we can create effect chains that contain combinations of effects with various numbers and orders. The labels are generated as a multi-hot vector in length 13, each element indicates the presence of one effect. For example, an array $[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ indicates an effect chain with overdrive, chorus, and high-reduct EQ. The labels are used as the ground truth during training and evaluation. We further split the effects into 6 different groups. Effects within a group are mutually exclusive: only one effect in the group is chosen in one effect chain. This is to avoid cancellation, such as applying Low boost and Low reduct at the same time. Table 2 shows the grouping.

Table 2: *Effect groups.*

Group number	Effect name
1	Overdrive
	Distortion
2	Chorus
	Flanger
	Phaser
	Tremolo
3	reverb
4	Feedback delay
	Slapback delay
5	Low boost
	Low reduct
6	High boost
	High reduct

During generation, we can choose the number of groups to apply, and the pipeline iterates over all combinations of groups, and combinations of the effects. Let E_n denote the total number of effect chains when applying n groups of effects, G_k denote the number of effects in the k-th group, and C_k^n denote a k-combination with n elements, we can calculate E_n using equation 1.

$$E_n = \sum_{g_i \in \{C_6^n\}} \sum_{k \in g_i} C_{G_k}^1 \tag{1}$$

For example, when $n = 2$, it iterates over the combinations of two groups: (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (4, 5). Within each set of groups, the combinations of effects are also iterated. For example in set $(5, 6)$, the combinations are (*Low boost*, *High boost*), (*Low boost*, *High reduct*), (*Low reduct*, *High boost*), (*Low reduct*, *High reduct*).

Each 5-second clean sample are rendered with all combinations of effects. We choose to use $n = [1, 5]$ in our generation process, where $n = 1$ is the case of single effect recognition and $n = 5$ means only one group is not used. As a result, 221 combinations of effects, 442,884 samples from `GuitarSet` and 143,640 samples from `IDMT-SMT-Guitar` are created. We further split `GuitarSet` samples into a training split of 327,522 samples and a validation split of 115,362 samples. We made sure the samples from the same audio source are in the same split to avoid the 'album effect' [16]. The `IDMT-SMT-Guitar` samples are only used during evaluation.

4. EXPERIMENTAL SETUP

We form two different setups: using the rendered audio with the clean audio ('with_clean'), or using the rendered audio only ('no_clean'). The 'with_clean' setup is a simpler task, since the model can compare the difference between the two signals and ignore unrelated variables, such as the type of guitar or the recording settings. Additionally, to compare with previous works on single-effect recognition, a reduced dataset with only $n = 1$ is used as a simpler setup.

To compare with previous works that use hand-crafted features, we use a traditional *Multi-Layer Perceptron* (MLP) model with *Mel-Frequency Cepstral Coefficients* (MFCC) [17] as input. The MFCCs are extracted from audio samples at $44,100$Hz with $n\_mfcc = 20$, $n\_fft = 4096$ and $hop\_length = 2048$. The MLP has three hidden layers with $hidden\_dim = 4096, 512, 13$ for each layer, and a ReLU function after each layer. The input shape of MLP is $2 \times 20 \times N_f$ for 'with_clean', and $20 \times N_f$ for 'no_clean', where $N_f = 108$ is the number of frames. As for the DC coefficient in MFCC, since it represents the overall volume of the audio, the MLP should be able to learn by itself whether the volume information is related to output labels.

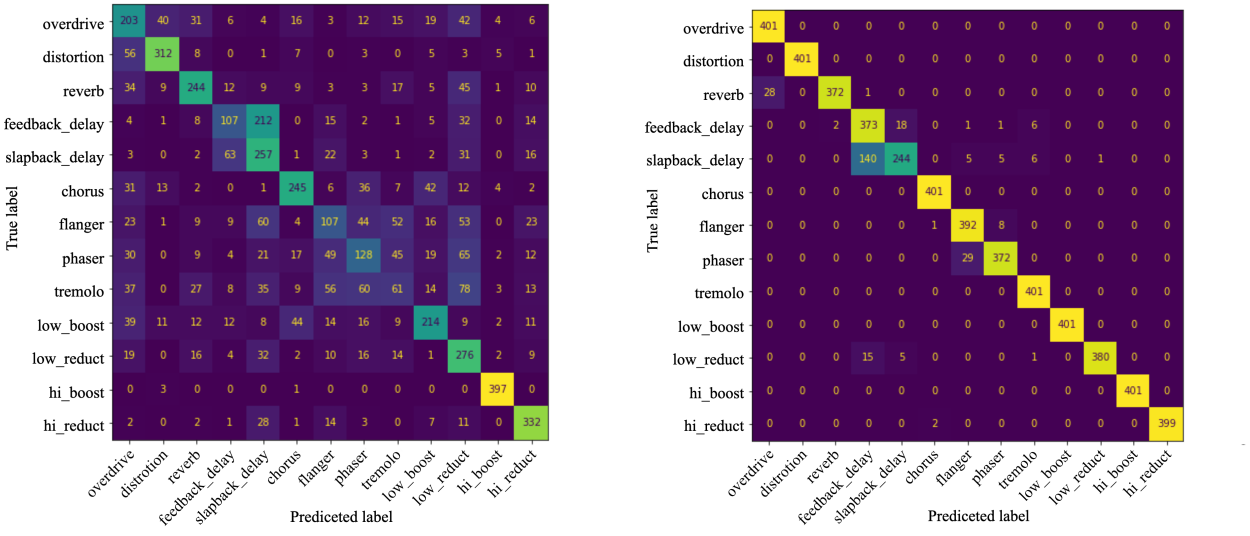
On the other hand, three newly developed Neural Network models are used: the *ResNet* model [7], the *CRNN* model [9] and the *Sample-Level CNN* model [10]. We use Mel spectrogram in dB scale as the input feature map for *ResNet* and *CRNN*, extracted from audio samples at $44,100$Hz with $n\_fft = 2048$, $n\_mels = 128$, $hop\_length = 1024$. The shape of one spectrogram is $(128, 216)$, with each time frame containing the information of 23.2ms. For the 'no_clean' task, the number of channels simply equals 1, while for 'with_clean' it becomes 2. The *Sample-level CNN* model uses raw audio downsampled to $22,050$Hz as input. The kernels and layers are slightly modified to suit the change of input and output size of our task.

The Models are built and trained using the `PyTorch` library [18]. For single-effect recognition, Categorical Cross-Entropy loss is used, while Binary Cross-Entropy loss is used for multi-effect recognition. Model weights are updated using Adam optimizer [19] with a learning rate of 0.001 for a maximum of 500 epochs. The batch size is set to 64 for baseline, *ResNet* and *CRNN*, and 16 for *sample-level CNN* due to memory limitation. The training set is shuffled before each epoch, but the random seed is fixed. An early stopping mechanism is performed if the validation loss over `Guitarset` validation split does not decrease for 2.5 epochs. For inference, the threshold of 0.5 is used. A computer with 16G RAM and one NVIDIA 2060 super GPU is used for training.

5. EVALUATION

5.1. Single-Effect Results

Table 3 shows the F_1 scores of the baseline model and *resnet18* model under the two setups. The *resnet18* model has outperformed the baseline model in both setups. For the single effect setup, the *'with_clean'* setup is relatively simple and both models achieved good performance. However, when the clean signal is removed,



(a) *no_clean setup.*

(b) *with_clean setup.*

Figure 2: *Single-effect confusion matrices of the baseline model under two setups, both on* `GuitarSet` *validation split.*

the baseline model has a significant performance drop, while *resnet18* can still achieve a score above 0.9.

Table 3: *Result of single effect recognition, evaluated on* `GuitarSet` *test split.*

Model	Setup	F_1 score
baseline	with_clean	0.956
resnet18	with_clean	0.999
baseline	no_clean	0.742
resnet18	no_clean	0.924

Figure 2 shows the single-effect confusion matrices of the baseline model under two setups. The characteristics of each effect class are different since they have different mechanisms. In the *'no_clean'* setup, *overdrive* and *distortion* are often confused, the reason might be that we actually used the same `SoX` function, only with different parameters. Since the data samples have different loudness levels, the clipping ratio varies according to the sample's own loudness level, which might cause the confusion. The *feedback delay* and *slapback delay* are similar, while the former has a feedback loop that also delays the previous outputs. More samples of *feedback delay* are classified as *slapback delay*, while fewer *slapback delay* are classified as *feedback delay*. The reason might be for samples without explicit note onsets, the differences between these two delays are too subtle for *MFCC*s to catch. Another confused group is *flanger*, *phaser*, and *tremolo*. Interestingly, it is the *tremolo* but not the *chorus* that is misclassified, since *tremolo* is amplitude modulation while *chorus*, *flanger* and *phaser* are all delay-based modulation. Since the baseline model is a plain MLP with no inductive bias for shift invariance, it did not capture the difference between delay modulation and amplitude modulation, but instead captured the difference in modulation patterns. In the `pysox` implementations, the Low Frequency Oscillators (LFO) are sinusoids in *flanger*, *phaser* and *tremolo*, while for *chorus* the LFO shape of each voice is randomly chosen between triangular and sinusoidal.

With the help of the clean signal, the model gained better performances in most of these groups. The model performed perfectly on *overdrive* and *distortion* when it knows the original loudness level of the signal. The volume information also helps to classify *tremolo* successfully from the other modulations. However, the model still struggles with *feedback delay* and *slapback delay*, which probably indicates that the MFCCs cannot catch such subtle timbre differences. There are still errors between *flanger* and *phaser*, which is more difficult when the audio sample only contains short impulses.

5.2. Multi-Effect Results

For a multi-label classification problem, we use both micro F_1 score and macro F_1 score as our evaluation metric. Figure 3 shows the benchmark of four models under two setups, two datasets. Among all the setups, the *resnet18* and *CRNN* model outperforms the baseline model under every setup, while the sample-level CNN only achieved a better micro F_1 score in one setup. The difference between micro F_1 and macro F_1 is relatively small for all models except sample-level CNN, which indicates that the performance of sample-level CNN is imbalanced between classes.

We can see a performance drop on `IDMT-SMT-Guitar` compared to the `GuitarSet` validation split. This indicates that even though the 'album effect' is avoided when splitting *GuitarSet*, the two splits are still inherently correlated. The F_1 scores of *CRNN* and *resnet18* on `Guitarset` validation split even reached 0.999 under 'with_clean' setup. Therefore, using `IDMT-SMT-Guitar` as the evaluation dataset is more meaningful.

In the hardest and most practical setup (`IDMT-SMT-Guitar`, 'no_clean'), the *resnet18* model achieved the best performance, while *CRNN* is slightly below. The other two models were able to solve the problem in the simpler setups, but failed in this practical setup. However, although the performance of *resnet18* and *CRNN* are similar in the hardest setup, we noticed that *resnet18* converges

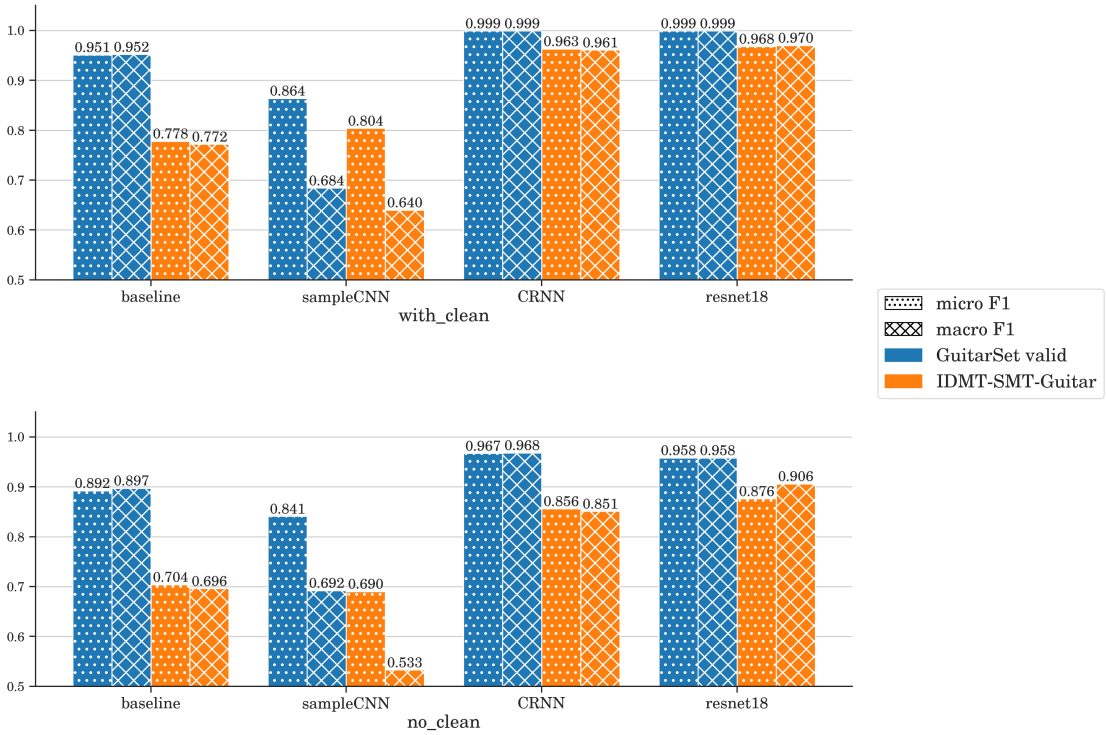


Figure 3: *Model benchmark of multi-effect recognition.*

much faster than *CRNN* during training.

Although the performance on `GuitarSet` validation split cannot directly represent the model's actual performance, we can analyze the robustness of models by comparing their differences in performance among two datasets. In both setups, the baseline model has the biggest performance drop on `IDMT-SMT-Guitar` dataset, while *resnet18* achieves the best robustness among the four models. All models had a bigger drop in 'no_clean' since the task is more difficult.

5.2.1. *Per-class analysis*

We used the best performing model, *resnet18* to analyze the result for each effect class. Figure 4 shows the confusion matrices of *resnet18* model under two setups. The performance under 'no_clean' setup is acceptable in most of the classes, except *overdrive* and *distortion*. The model achieved F_1 scores around 0.93 for these two classes on `GuitarSet` validation split, but dropped significantly on `IDMT-SMT-Guitar` while other classes achieved the same level of performance among two datasets. The reason might be that we used the same `SoX` implementation for these two classes, only with different gain levels. When testing on another dataset with a different distribution of loudness levels, the distribution compression rate also varies significantly. On the other hand, when clean audio is given to the model in 'with_clean' setup, it is easier to learn the relationship between input and output gain, and the model achieved a better performance on the two classes. Other classes benefit from the additional information as well.

5.3. Ablation study

We performed an ablation study on *resnet* by removing groups of convolutional layers at the output end. Since *resnet18* already performed well under both setups and both datasets, we want to investigate the minimum number of layers required to get such performance. The ablation models are trained using the same configuration as described in Section 4. For each ablation model, the last group of convolutional layers is removed, and the number of layers decreases by 4. Metrics are evaluated on `IDMT-SMT-Guitar` dataset.

Table 4: *Model complexity and performance for resnet ablations. Metrics are evaluated on* `IDMT-SMT-Guitar`.

Model	Setup	Number of parameters (k)	Micro F_1	Macro F_1
resnet18	with_clean	714.7	0.968	0.970
resnet14	with_clean	188.3	0.963	0.955
resnet10	with_clean	56.2	0.958	0.950
resnet6	with_clean	34.4	0.926	0.917
resnet18	no_clean	714.3	0.876	0.906
resnet14	no_clean	187.9	0.848	0.832
resnet10	no_clean	55.6	0.860	0.844
resnet6	no_clean	34.0	0.830	0.811

Table 4 shows the complexity and performance of the ablation models. The performance drop has a nonlinear relationship with respect to the number of parameters. We see that *resnet14* and *resnet10* have similar performance, and *resnet10* even slightly outperformed *resnet14* under 'no_clean' setup. On the other hand,

(a) *'with_clean'* setup

(b) *'no_clean'* setup.

Figure 4: *Confusion matrices and class-wise F1 scores of resnet18, multi-effect* `IDMT-SMT-Guitar`.

resnet6 has a performance drop of about 0.03 under both setups. Generally, the performance under 'no_clean' setup is a fair trade-off between complexity and performance. For the 'with_clean' setup, the smallest model is still able to perform relatively well, since the task is much simpler with the clean signal.

6. CONCLUSION

In this paper, we reconstructed the task of guitar effect recognition to make it closer to the actual use-case of guitar effect pedals. We created a workflow that renders arbitrary guitar recordings with all combinations of effects using `SoX`, which increases the variation compared to existing datasets. Secondly, we adapted novel Neural Network models to solve the multi-label classification task under two setups, 'with_clean' and 'no_clean'. The best performing model is modified from the *resnet18* model, achieving a micro F_1 of 0.876 and macro F_1 of 0.906 on an unseen dataset under 'no_clean' setup. Class-wise analysis indicates that the model is able to distinguish most of the effects except a small confusion on *overdrive* and *distortion*. An ablation study shows that the deep structure is necessary to achieve the performance, but a trade-off between complexity and performance is optional. Possible improvements include randomizing the effect order and parameters, or using better effect plugins, even real effect units.

7. REFERENCES

[1] Michael Stein, Jakob Abeßer, Christian Dittmar, and Gerald Schuller, "Automatic detection of audio effects in guitar and bass eecordings," *Journal of the Audio Engineering Society*, May 2010.

[2] Michael Stein, Jakob Abeßer, Christian Dittmar, and Gerald Schuller, "Automatic detection of multiple, cascaded audio effects in guitar recordings," in *Proc. Digital Audio Effects (DAFx-10)*. Graz, Austria, Sep. 2010, pp. 4–7.

[3] Maximilian Schmitt and Björn Schuller, "Recognising guitar effects - which acoustic features really matter?," in *INFORMATIK 2017*. Chemnitz, Germany, 2017, pp. 177–190.

[4] Henrik Jürgens, Reemt Hinrichs, and Jörn Ostermann, "Recognizing guitar effects and their parameter settings," in *Proc. Digital Audio Effects (DAFx-20)*. Vienna, Austria, Sep. 2020.

[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, Jul. 2021.

[6] Christopher M. Bishop, *Neural networks for pattern recognition*, pp. 116–161, Oxford university press, 1995.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Nevada, USA, Jun. 2016, pp. 770–778.

[8] Dipjyoti Bisharad and Rabul Hussain Laskar, "Music genre recognition using residual neural networks," in *2019 IEEE Region 10 Conf. (TENCON)*. Kochi, India, 2019, pp. 2063–2068.

[9] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho, "Convolutional recurrent neural networks for music classification," in *IEEE Intl. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 2392–2396.

[10] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms," in *Proc. 14th Sound and Music Computing Conf. (SMC)*. Espoo, Finland, Jul. 2017.

[11] Taejun Kim, Jongpil Lee, and Juhan Nam, "Sample-level cnn architectures for music auto-tagging using raw waveforms," in *2018 IEEE Intl. Conf. Acoustics, Speech and Signal Processing (ICASSP)*. Calgary, Alberta, Canada, 2018, pp. 366–370.

[12] Qingyang Xi, Rachel M. Bittner, Johan Pauwels, Xuzhou Ye, and Juan Pablo Bello, "Guitarset: A dataset for guitar transcription.," in *Proc. Intl. Society for Music Information Retrieval Conf. (ISMIR)*. Suzhou, China, 2018.

[13] Christian Kehling, Jakob Abeßer, Christian Dittmar, and Gerald Schuller, "Automatic tablature transcription of electric guitar recordings by estimation of score- and instrument-related parameters," in *Proc. Digital Audio Effects (DAFx-14)*, 2014.

[14] Chris Bagwell and SoX contributers, "Sox: Sound exchange, the swiss army knife of sound processing," 1991.

[15] Rachel Bittner, Eric Humphrey, and Juan Bello, "Pysox: Leveraging the audio signal processing power of sox in python," in *Proc. Intl. Society for Music Information Retrieval Conf. (ISMIR) Late Breaking and Demo Papers*. New York City, USA, Aug. 2016.

[16] Brian Whitman, Gary Flake, and Steve Lawrence, "Artist detection in music with minnowmatch," in *Neural Networks for Signal Processing XI: Proc. 2001 IEEE Signal Processing Society Workshop*, 2001, pp. 559–568.

[17] Beth Logan, "Mel frequency cepstral coefficients for music modeling.," in *Proc. Intl. Symposium on Music Information Retrieval (ISMIR)*. Plymouth, Massachusetts, USA, 2000, vol. 270, p. 11.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, Dec. 2019, vol. 32.

[19] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.