A TINY MACHINE LEARNING CASE STUDY: BELA-BASED GUITAR DISTORTIONS

Marco Furio Colombo, Rebecca Superbo

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano
`[marcofurio.colombo, rebecca.superbo]@mail.polimi.it`
`https://github.com/FurioColombo/SeletectedTopics2023ForgerNFX`

ABSTRACT

This project presents the development and implementation of a real-time neural-effects processor meant for guitar, using the RTneural framework on the Bela platform. The obvious hardware limitations are the perfect cue to explore the integration of machine learning techniques in audio signal processing within the context of tiny machine learning. The price of inherently having serious resources limits in our embedded computing platform is however paid back by a latency low enough to be consistently not human perceivable. The proposed model is based on LSTM networks and performs time-domain filtering on the received audio inputs. The model is trained on audio samples of the same instrument recorded both with a clean tone and the distorted one. Two different pipelines provide suitable dataset formats ready-to-use for the model training, coping with the specific nature of the available data. The architecture and training methodology are thoroughly examined, with particular focus on the real-time performance, which is an essential condition to achieve and the integration of the trained model into the Bela platform is explained, with special focus on the software configuration and on the adjustments for hard real-time audio processing. Finally we implemented some performance evaluation testing and comparing the latency and sound quality, together with a specific test on Bela's performances. Objective metrics validate the effectiveness of the LSTM-based RTneural guitar effects processor on the Bela platform, underscoring its potential for real-time audio applications. One might state that the successful integration of our system with the Bela platform is a proof of the potential of tiny machine learning for innovative real-time audio processing systems that encourages the research and development of audio effects and other audio-related domains in such context.

Index Terms— BELA, Embedded systems, RTNeural, LSTM, Neural effects

1. INTRODUCTION

In recent years, the field of audio signal processing has witnessed a paradigm shift with the emergence of neural network-based techniques. Traditional audio effects processors have relied on digital signal processing (DSP) algorithms, but with the advent of deep learning, neural effects processors have proven to be a transformative innovation in the world of audio engineering.
Neural effects processors have demonstrated their capability in various applications, including speech synthesis, music generation, and virtual instrument modeling. By means of training neural networks on vast datasets of audio samples, these processors can capture intricate nuances and intricacies of sound, allowing for dynamic audio processing in real-time. An honorable mention is due to state of the art profilers produced by manufacturers like Kemper [1] and neural DSP [2], which have quickly gained popularity and have become a staple for many professional musicians. This has paved new ways for artists, producers, and audio enthusiasts to explore creative possibilities and push the boundaries of audio processing in ways unimaginable just a decade ago.

In the light of these recent developments, our project aims to be a proof of concept for the potential of neural audio effects in a hard real-time context and with limited computing resources. Our integration of neural network technology into the guitar effects domain allowed the creation of the replica of three real-time guitar distortion effects, based on real audio recordings of analog guitar pedals.

2. SYSTEM

Our computing system of choice is Bela [3], a platform built upon the BeagleBone Black [4], a single-board computer which features a 1GHz ARM Cortex-A8 processor and 512MB of RAM. Bela runs a custom audio processing environment based on the Xenomai real-time Linux extensions, allowing code to run in hard real-time, bypassing the entire operating system to go straight to the hardware. This results in the capability of producing latency as low as 1 millisecond from audio in to audio out, when sampling a small number and performance not being affected by other system load [5].

The audio processing logic is carried out in the time domain by a simple deep learning model developed using python and pytorch [6]. The dimensions of such model are dependant on its performance on the limited resources of a Bela and therefore we are constrained in the tiny machine learning domain. The foundation of our model lies is the LSTM (Long Short-Term Memory), a type of recurrent neural network able to capture temporal dependencies in sequential data.

Due to the lack of consistent datasets suited for our purposes, two different methods for generating them are proposed: EFGX and Fragments. The user is free to choose considering pros and cons of each method.

Through the RTNeural library [7] employed in the cross-compilation phase, we are able to adapt our neural effects processor, originally developed using Python, to operate seamlessly within the C++ ecosystem of Bela.

The user interface is coded in a Javascript file, aiming at a captivating and realistic experience for our embedded system.

Three kinds of technical evaluations are proposed and discussed, one regarding the performance of the system, and two considering the dataset selection and training process focusing on the Mean-Squared Error (MSE).

3. IMPLEMENTATION

Our system is capable of emulating guitar pedal effects, specifically designed for distortion effects. At its core lies a deep learning model that takes a clean guitar signal as input and produces a wet sound as output. It is not a conditional model, thus it does not provide the option to adjust the level of distortion or modify any of its parameters.

3.1. Model Architecture

The heart of our LSTM-based RTneural guitar effects processor lies in its parametric neural network model architecture, designed to extract the audio features from a curated dataset and generate faithful effects in real-time.
The architecture is composed of two components: an LSTM layer and a fully connected layer. Its configuration parameters are contained in a configuration file that allows the customization of the network model and training. Besides the model dimensions, training sample rate and even type of network, it is worth noting that we are able to choose between two different input sizes for the model, 1 and 16. This corresponds to the minimum size of the block that can be processed in inference, thus determining the minimum Bela block size, which is the amount of sample processed together, and thus the latency.

The LSTM layer forms the core of the model, enabling the capture of temporal dependencies within sequential audio data. This layer excels in handling time-series data, making it a natural choice for audio signal processing. Each LSTM unit processes input sequences and maintains an internal state, allowing it to remember relevant information over varying time intervals. This characteristic proves essential for understanding the context and dynamics of audio signals, which is critical in generating authentic and responsive guitar effects.

Following the LSTM layer, the fully connected layer is responsible for translating the embedding produced by the LSTM layer back into the sample domain and therefore outputting audio samples ready to be reproduced. This layer comprises densely connected neurons that take the LSTM's outputs as input and apply learned weights and biases to transform the data. The fully connected layer performs the final transformations that generate the desired audio effects, shaping the processed audio signal according to the learned patterns and characteristics.

The simplicity of this kind is enforced by the small computational power and the constraint of hard real-time inference, nonetheless it has proved valid and offers satisfies our prerequisites. The LSTM layer excels at capturing intricate audio features, while the fully connected layer effectively translates these features into expressive guitar effects. This straightforward design enhances computational efficiency, enabling the model to run seamlessly on the Bela platform with minimal latency. Furthermore, the model's relatively low complexity allows for easy interpretation and potential future enhancements.

To ensure the fidelity of the generated guitar effects and the accurate reproduction of input audio characteristics, we employed a comprehensive loss function [8] that amalgamates three distinct power matching metrics. The primary objective of this loss function is to guide the training process, guiding the model to produce effects that not only sound realistic but also maintain the inherent power distribution of the input audio.

3.2. Dataset

For the model to learn the transformation between two signals, we need to find or create a dataset of appropriate signals from which to learn.

Unfortunately, there isn't an extensive supply of suitable datasets tailored to our purpose. Hence, we've developed two distinct methods to generate datasets for our training.

The first method uses recordings of individual guitar notes both in their clean and altered states for various effects and guitar pickups. Our framework processes files from this dataset, making them usable with PyTorch for network training. This was inspired by the publicly available EGFxSet dataset [9], which we extensively used in our training runs. This dataset offers the advantage of consistency by comparing individual notes, resulting in very clean audio. However, it falls short in containing complete musical phrases, chords or other uses of the instrument that the model could potentially learn from.

The second method involves data extraction based on a previously provided and reviewed code. This method fits the needs for those who wish to create their own dataset in a short time, to circumvent the lack of well-constructed online datasets. It takes a clean track and its corresponding wet track (for the same notes) and breaks them into equal-duration fragments. These fragments populate a dataset that can be used for training. The advantage here lies in its comprehensiveness, including both clean notes and phrases/passages with variations. However, the potential downside is its theoretically greater complexity to learn due to the diverse and potentially less meaningful data it contains (e.g., silences or moments of noise).

3.3. Cross-compilation and Inference

Bela is an embedded computing platform that provides ultra-low latency, high quality audio, analog and digital I/O in a tiny self-contained package. It can be easily embedded into a huge range of applications, and in our case, as the platform for the inference. While the model's training is carried out in python, the language is not really fit for performance optimization and hard real-time applications and is not supported on Bela anyway. On the other hand C++ is perfectly fitted for our purposes and is the original language for Bela, also remaining the core of Bela language support. Therefore a transition from the Python-based development environment to C++ (onto which Bela is based) is needed; This involves re-implementing the core components of the model architecture and integrating them into the Bela ecosystem.

In order to make the development of the inference code more sustainable and fast, we rely on a cross-compilation framework based on previous works published by Bela enthusiasts [10] [11]. This allows us to develop all our code outside the board, speeding up dramatically the build times and the control over the development environment. The development can be theoretically carried out in any linux environment, but we only tested it extensively using Debian. After developing and building the executable, it is possible to simply copy it on the board and run it.

We employ a lightweight neural network inferencing engine written in C++, *RTNeural*. This library is designed with the intention of being used in real-time systems, specifically real-time audio processing as it is capable of taking a neural network that has already been trained, loading the weights from that network, and running inference. In this regard, our implementation allows for choosing different Bela block sizes together with different block size for the models obtained after the training process. The relationship between them is a crucial feature in the evaluation part of our project.

3.4. GUI

A crucial aspect of the user experience is the ability to control and customize the guitar effects in real-time. To achieve this, we have developed a user-friendly Graphical User Interface (GUI) that empowers guitarists to interact with and manipulate the effects effortlessly. It is integrated into the Bela platform, enabling users to seamlessly control the neural effects processor using a web browser, and designed using JavaScript, which establishes a connection between the web browser and the Bela platform.

The GUI presents a collection of buttons that enable users to control various parameters of the effects processor:

- Clean Button: This button deactivates all effects, producing a clean, unprocessed guitar sound.
- Effect Buttons: Three dedicated buttons correspond to the different neural effects. Guitarists can select and activate an effect of their choice using these buttons.
- Model Block Size Switch: A switch button allows users to toggle between two model block sizes: 1 and 16. This feature empowers guitarists to explore the trade-offs between lower latency (block size 1) and higher efficiency (block size 16).
- Effects On/Off Button: This button toggles the overall effects processing on and off, giving guitarists the flexibility to switch between the unprocessed and processed sound during performance or practice.
- Effect Selection Button: An additional button enables users to cycle through the available effects. This feature allows for creative exploration and rapid switching between effects to adapt to the musical context.

The GUI is accessible through a web browser, facilitated by the Bela platform's support for interfacing via Google Chrome. Users can connect their devices to the Bela board's IP address, accessing the GUI in real-time.

User selections made in the GUI are communicated to the Bela platform's core script, *render.cpp*, using the *sketch.js* file. The script deciphers the user's choices and applies the corresponding control signals to the neural effects processor. This seamless interaction enables immediate and intuitive control over the effects, directly from the user's web browser.

4. EVALUATION AND DISCUSSION

4.1. Performance

We analyzed the variations in the performance of our system as it changes depending on two parameters: the Bela block size and the model's block size. The Bela block size is the number of audio samples processed simultaneously by the machine. By increasing its size we trade off a slightly increased delay in the inference for a decreased cost of the computation's overheads. Since we are operating in a hard real-time context, the largest Bela block analyzed is of 256 samples.
The model's block size is the number of audio samples that together create the chunks of data used as the model's inputs. A block size of 1 indicates that the model processes audio samples one at a time, while a block size of 16 allows processing a group of 16 consecutive audio samples simultaneously.

Model block size = 1					
LSTM maximum size	6	6	6	6	8
BELA block size	16	32	64	128	256

Table 1: Table 1 for the model with a block size of 1

Model block size = 16					
LSTM maximum size	64	92	128	192	256
BELA block size	16	32	64	128	256

Table 2: Table 2 for the model with a block size of 16

Then for each model block size, we explored various LSTM sizes that could be correctly processed by different block sizes of the Bela platform, ranging from 16 to 256. The LSTM size is a crucial parameter affecting the model's capability to handle the audio data efficiently while minimizing latency and preserving sound quality.
As for the objective metric, we consider the processing efficiency, i.e. the processor's ability to handle real-time audio processing with minimal computational overhead.
The evaluation results on performance clearly indicated that the choice of model block size and LSTM architecture significantly impacted the performance of our embedded system.
For the model with a block size of 1, our evaluations revealed that LSTM sizes of 6 worked optimally with Bela block sizes from 6 to 128. The largest Bela block analyzed is of 256 samples, allowing for a small improvement in terms of LSTM size, while still ensuring robust and high-fidelity audio processing.
On the other hand, a model block size of 16 enables a substantial increase in the maximum available network size, resulting in a clear improvement in terms of computing performance. However, from the output audio standpoint, the quality appears to be lower. This issue might be addressable through training on larger datasets, but unfortunately, due to time and power constraints in the available machines, generating such datasets is not feasible.

4.2. Datasets

Comparing the MSE values between EGFxDataset and Fragments sheds light on the strengths and limitations of each dataset. Lower MSE values signify a more faithful reproduction of the effects, indicating that the model effectively learns the desired transformations. On the other hand, higher MSE values might suggest challenges in accurately capturing certain nuances or dynamics.

The MSE evaluation results carry profound implications for the training process. They guide the selection of datasets that yield more accurate and musically relevant audio transformations. By understanding which dataset aligns better with the target effects, we can tailor the training regime to emphasize the strengths of the chosen dataset while addressing any potential shortcomings.

The evaluation of MSE between EGFxDataset and Fragments provides a quantitative measure of the effectiveness of our training datasets, the results here reported in Figure 1 clearly show how the EGFx dataset grants a lower MSE, indicating that the model's performance is better in terms of capturing the patterns and relationships within the data and is more accurate in its predictions with a greater ability to generalize to new, unseen data.

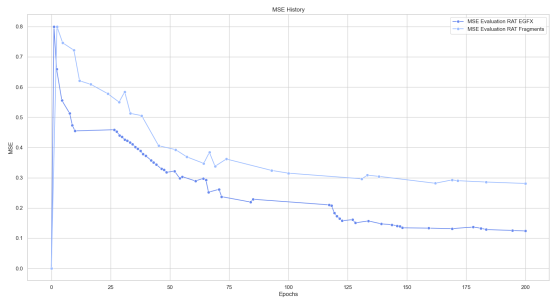

Figure 1: MSE evaluation between EGFx and Fragments datasets.

4.3. Training

For the evaluation of the training process, we conduct a series of comparisons using LSTM networks with the same dimensions and fixed parameters. Even in this evaluation, we focus on two specific scenarios, one with a model block size of 1 and the other with a block size of 16.

In both scenarios, we train three LSTM networks with the same size that was found in the previous performance evaluation, one for each dataset. This process allows us to determine the optimal LSTM size that strikes a balance between expressiveness and computational efficiency for real-time processing.

With a fixed block size, the system gains ability to handle larger audio chunks efficiently, meaning that the machine can handle bigger and more complex models.

To ensure consistency and reliability in our evaluations, we maintain the maximum number of epochs fixed when comparing iterations of training of the same model over different datasets.

The evaluation process involves measuring the Mean Squared Error (MSE) for each trained model in order to quantify the discrepancy between the predicted and target audio signals. The MSE values are visualized using a Line Plot, that allows us to observe the convergence of the MSE over the fixed number of epochs for each dataset and block size scenario.

The results obtained from the Line Plots reaffirmed the importance of selecting appropriate LSTM sizes for different model block sizes, as expected.

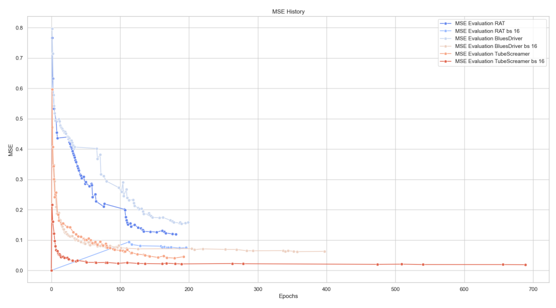

Figure 2: MSE evaluation within the EGFx dataset.

5. CONCLUSION AND FUTURE WORKS

In conclusion, our user study has yielded a multi-effect distortion processor that behaves quite fine both in sonic performance and visual appeal. Even employing compact neural network architectures, we were able to obtain satisfying reproductions of the original analog distortion effects. Moreover, the achieved outcomes reveal the system's ability to fully meet tight real-time requirements, ensuring seamless audio processing using a compact system.
Looking ahead, the rise of TinyML suggests a transformative future for DSP. Currently, the associated costs remain within accessible bounds, and computing platforms akin to the one employed in this project are both economically viable and widely accessible. This underscores the potential of DSP democratization through neural networks, extending its capabilities to broader domains.

Our project leaves open perspectives in terms of creativity; first of all, it could be turned into a physical interface, reminiscent of traditional guitar pedals. With Bela's capabilities, we could bridge the digital and physical aspects of music-making. Moreover, there's the potential for further advancement in terms of model capabilities: we have faced issues in trying to reproduce time-varying effects. A natural development would be to implement more complex models involving different types of neural layers.

The integration of these emerging techniques with traditional DSP approaches introduces an exciting avenue for exploration. This convergence can potentially lead to innovative solutions that leverage neural networks' adaptability and DSP's established principles. This hybrid approach holds promise for unlocking new dimensions of audio processing and creative expression.

6. REFERENCES

[1] https://www.kemper-amps.com/.

[2] https://neuraldsp.com/quad-cortex.

[3] https://bela.io/.

[4] https://www.beagleboard.org/boards/beaglebone-black.

[5] A. McPherson, R. Jack, and G. Moro, "Action-sound latency: Are our tools fast enough?" in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Brisbane, Australia: Queensland Conservatorium Griffith University, 2016, pp. 20–25. [Online]. Available: http://www.nime.org/proceedings/2016/nime2016_paper0005.pdf

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[7] https://github.com/jatinchowdhury18/RTNeural/.

[8] https://github.com/Alec-Wright/CoreAudioML/blob/bad9469f94a2fa63a50d70ff75f5eff2208ba03f/training.py/.

[9] https://paperswithcode.com/dataset/egfxset/.

[10] https://github.com/thetechnobear/xcBela/.

[11] https://github.com/maxmarsc/xc-bela-cmake.