ON THE CHALLENGES OF EMBEDDED REAL-TIME MUSIC INFORMATION RETRIEVAL

Domenico Stefani and Luca Turchet

Department of Information Engineering and Computer Science
University of Trento
Trento, Italy
domenico.stefani@unitn.it | luca.turchet@unitn.it

ABSTRACT

Real-time applications of Music Information Retrieval (MIR) have been gaining interest as of recently. However, as deep learning becomes more and more ubiquitous for music analysis tasks, several challenges and limitations need to be overcome to deliver accurate and quick real-time MIR systems. In addition, modern embedded computers offer great potential for compact systems that use MIR algorithms, such as digital musical instruments. However, embedded computing hardware is generally resource constrained, posing additional limitations. In this paper, we identify and discuss the challenges and limitations of embedded real-time MIR. Furthermore, we discuss potential solutions to these challenges, and demonstrate their validity by presenting an embedded real-time classifier of expressive acoustic guitar techniques. The classifier achieved 99.2% accuracy in distinguishing pitched and percussive techniques and a 99.1% average accuracy in distinguishing four distinct percussive techniques with a fifth class for pitched sounds. The full classification task is a considerably more complex learning problem, with our preliminary results reaching only 56.5% accuracy. The results were produced with an average latency of 30.7 ms.

1. INTRODUCTION

Music Information Retrieval (MIR) is a field of research that focuses on the analysis and extraction of information from music. Prominent examples of MIR tasks include beat-detection [1], onset detection [2], music transcription [3], and genre classification [4]. To date, research in this area has mainly focused on offline methods, which operate on large datasets and do not have tight constraints on execution time. However, the sub-field of *real-time Music Information Retrieval* (rt-MIR) has been gaining more interest recently, since it can be used for the development of music performance tools such as digital musical instruments [5] [6], including smart musical instruments [7].

Rt-MIR offers a number of challenges and constraints that are not present in the offline context, as real-time systems pose strict requirements on the execution time of algorithms and the latency of these systems [8]. These requirements are particularly stressed by the complexity of deep learning, which can deliver highly accurate results at the expense of computational resources. Moreover, embedded devices and single-board computers have become a powerful tool for compact learning and music systems [9] [10] [11], but their limited computational resources pose additional challenges for rt-MIR.

In this paper, we identify and discuss the main challenges that can be found when developing embedded rt-MIR systems dealing with acoustic signals. We also propose various solutions to these challenges. Finally, we demonstrate the validity of some of the proposed solutions by implementing an embedded real-time expressive technique classifier for the acoustic guitar. In particular, we focus on CPU-based embedded hardware such as single-board computers, which can run an operating system. This excludes simpler devices such as microcontrollers.

Our implementation of an expressive technique classifier can be compared to the work of Reboursiere et al. [12], which consisted of a series of algorithms to track in real-time a similar set of expressive guitar techniques. However, despite the high accuracy obtained by the authors on polyphonic performances, the different techniques were tracked with separate algorithms and tested separately because of processing power issues. Additionally, this work did not address an embedded implementation of the system. An earlier report by the authors [13] describes an embedded implementation with a Field Programmable Gate Array (FPGA), but it was presented as an incomplete project and the algorithms were tested separately and offline (not in real-time). Additionally, differently from our implementation, these approaches do not address percussive fingerstyle techniques on the acoustic guitar. Real-time classification of two percussive techniques was tackled in a preliminary manner by Martelloni et al. [14]. However, the authors did not propose an implementation for embedded systems, nor did they provide accuracy or latency measurements. Furthermore, the authors reported that the accuracy was rather low, causing "discomfort" when using the system to trigger audio samples.

The reminder of the paper is organized as follows. Section 2 describes the challenges and limitations of embedded rt-MIR. Section 3 presents solutions to the aforementioned challenges. Then, in Section 4 we present an embedded real-time classifier for expressive playing technique recognition on the acoustic guitar. Finally, we draw our conclusions in Section 5

2. CHALLENGES OF EMBEDDED REAL-TIME MUSIC INFORMATION RETRIEVAL

This section describes the main challenges behind designing and implementing an embedded rt-MIR system that use acoustic signals as input. Such challenges have been identified as a result of our research in the field of smart musical instruments [7], a class of digital musical instruments based on real-time embedded systems, which can run rt-MIR algorithms for different purposes, such as real-time expressive playing technique retrieval. MIR algorithms

that operate on input data other than acoustic signals (e.g., symbolic music representation) are outside of the scope of this study. The identified challenges, described in detail below, are as follows:

1. Availability of only causal information;
2. Tradeoff between accuracy and latency;
3. Audio processing deadlines and real-time-safe programming rules;
4. Embedded hardware and software limitations.

2.1. Availability of only causal information

The main difference between offline and rt-MIR systems is the possibility of analyzing the entire input signal at any point in time. For offline systems, this means that more information can often be gathered by collecting large analysis windows around events of interest in the input signal. Moreover, offline MIR algorithms can move forward and backward along the entire input audio signal. On the contrary, rt-MIR systems can only process past and current inputs. For example, the classification of musical properties of events in a signal (e.g., instrument detection) can only happen with a non-zero delay from such events (i.e., latency) which is partly due to need of the classification algorithm to wait to collect a signal window to analyze. Since more input information about a musical event in the signal can potentially yield higher classification accuracies, similar rt-MIR systems must be tuned to adjust the tradeoff between their accuracy and the latency with which the results are produced. The same holds for non-classification rt-MIR systems (e.g., regression [15]) and more general audio tasks (e.g., environmental sound recognition [9]). This will be discussed in detail in part of the next section.

Moreover, some methods or algorithms used in MIR can only be applied to offline contexts, such as Bi-directional Recurrent Neural Networks (BiRNNs). This class of neural networks has been used successfully for tasks such as offline music-genre classification [16] and onset detection [17], but these are designed to simultaneously use information from both the past and the future to operate, which renders them unusable for rt-MIR.

In addition, some offline algorithms cannot be used in real-time systems simply because they require the entire audio signal. This is, for example, the case of the whitening process, which consists in dividing the magnitude of each bin of the Short-time Fourier transform by the all-time maximum value for that bin. Despite improving the performance of onset detectors in many cases [18], whitening uses non-causal information and cannot be used for real-time onset detectors. An overview of the real-time capabilities of some of the onset detection methods mentioned was presented by Bock et al. [19].

2.2. Tradeoff between accuracy and latency

In rt-MIR systems, a tradeoff is established between the accuracy of the results of any analysis and the latency with which these are produced. This is due to two main reasons: the availability of only past information (causal) and the relation between the accuracy and execution time of many analysis algorithms, especially machine learning and deep learning techniques.

As introduced in the previous section, rt-MIR systems can only process their past and current inputs. However, their computation can be delayed in time to collect a wider signal window, which could contain more information about the part of interest of the input signal. For many tasks, this can directly translate to an increase in the accuracy of the results of the analysis. For example, delaying the analysis of a sound in a pitch tracking or sound event recognition system can allow a rt-MIR algorithm to consider a larger part of the temporal envelope and spectral content of the sound, thus increasing the quality of the features that can be extracted and helping to discard potential false positives. At the same time, delaying the analysis of the signal introduces some latency between the input and when the results are produced, which can only be tolerated to a specific degree for different tasks and applications. For example, a system that reacts to a sound by producing new sounds will require a latency that is about 30 ms or lower, since complex tones that are separated by less than such a delay may be perceived by the human hearing system as simultaneous [20].

Apart from delays that can be introduced on purpose, the execution time of rt-MIR algorithms also affects the total latency with which the results of a real-time system are produced. This is particularly critical with modern machine learning and deep learning algorithms, which are often designed to achieve the best accuracy in contexts where the execution time is not bounded (i.e., offline MIR). For most of these algorithms, the execution time can be effectively reduced, but it will reduce the quality of the results. For example, the execution time of inference with a K-nearest neighbors (KNN) classifier can be high with a large number of training samples, but it can be reduced by removing an arbitrary number of samples [21]. However, this operation will effectively reduce the amount of information that describes the output classes. On the contrary, reducing training data for deep learning algorithms will have no effect on inference time, but this can be reduced greatly by either decreasing the width and depth of neural network models or reducing the numeric precision of the weights of a network (weight quantization). Any of these operations will necessarily reduce the quality of the results.

2.3. Audio processing deadlines and real-time-safe programming rules

Real-time audio software handles the input and output of the audio signals with buffers. Audio buffers, which have a predefined size in samples, are delivered to the software at a rate that depends on the audio sampling rate and the predefined buffer size. As a consequence, these systems have a limited amount of time to process the input buffers before the succeeding buffer is delivered in input. This limited "time budget" is particularly relevant for systems that also have to deliver an output audio signal, since any failure in performing every computation inside this time slot will reflect on the output in the form of audible glitches. For rt-MIR systems that operate on an input audio signal, at least part of the analysis computations must be performed at the same rate as the audio processing. As a result, these computations will add to the time used out of the available budget. This also holds for rt-MIR systems that operate on input audio signals.

Lastly, real-time audio programming requires developers to use only operations that are guaranteed to execute in a given amount of time to complete (i.e., real-time-safe operations[1] [22]), which prevents the aforementioned issues. This requires developers to avoid dynamic memory allocation, locking operations, and more instructions that can force the real-time thread to wait on lower priority operations. These rules also apply to any external library

[1] http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing

used, which can be a problem with code designed for offline use (e.g., offline MIR and libraries for deep learning inference).

2.4. Embedded hardware and software limitations

The performance of any rt-MIR system is dependent on the limitations of the computing hardware used, along with the low-level software (e.g., OS) of its target computing device. While modern PCs are generally powerful and fitted with hardware for high throughput operations (i.e., GPUs), embedded computers tend to be resource-constrained platforms that offer many challenges for real-time audio and rt-MIR applications.

First of all, deep learning algorithms have almost completely replaced traditional machine learning and simpler heuristics in many MIR areas [3] [17] [23] [24], but they tend to be more computationally complex. Furthermore, embedded computers are generally resource-constrained devices when compared to general purpose PCs. Other than limited CPU speeds, core count, RAM amount, and speed, very few embedded computers are provided with GPUs or similar acceleration hardware for deep learning inference, such as Tensor Processing Units (TPUs). Moreover, while it is possible to collect an audio signal on-site and process it in the cloud (on a powerful server), this introduces high communication latencies that are not desirable in a real-time system.

3. POTENTIAL SOLUTIONS AND VIABLE TRADEOFFS

3.1. Availability of only causal information

In Section 2.1 we discussed the limitation imposed to rt-MIR systems due to the availability of only past and current input information. In some cases, this limitation requires developers to devise entirely different methods to those used in offline contexts. However, if the target task of a rt-MIR system allows for a tolerable degree of latency, the analysis of the signal can be delayed as much as possible, so to give as much information as possible to the processing algorithm about the input signal. For example, for the expressive technique recognition algorithm presented in Section 4, the feature extraction window was set to be as large as possible, while fitting in the maximum target latency along with the delays introduced by the other components of the classification algorithm.

However, some algorithms cannot be used in real-time contexts by definition. These include techniques that need to collect information about the entire signal beforehand, such as whitening (see Section 2.1). In many cases, such as for whitening, these operations can be replaced by their approximate versions, which only rely on the part of the signal gathered up to each point of the analysis. This is the case of the adaptive whitening technique for onset detection presented by Stowell et al. [18].

Similarly, some operations in neural networks can only be used for offline MIR. This is the case of the bidirectional information flow in BiRNNs [16] [17], which must be replaced with unidirectional Recurrent Neural Networks (RNNs). A similar approach was used by Bock et al. [23], who presented a successful real-time onset detector, which was adapted from a previously developed approach for offline contexts.

3.2. Tradeoff between accuracy and latency

As mentioned in the previous sections, signal analysis can be delayed up to a maximum tolerable latency. This is in itself a tradeoff, since it involves forgoing a potentially high accuracy to achieve a specific maximum tolerable latency.

In the case of a deep classifier, this can be obtained by making a neural network shallower, narrower, or reducing the precision of the neuron weights (i.e., weight quantization). The first two methods will require retraining the neural network and will reflect directly on the execution time, since they reduce the number of computations that need to be performed. In our classifier, we tuned the size of the classifier network to fit the target latency (see Section 4.3). Differently, weight quantization is performed on trained models, and it consists in reducing the precision of floating-point weights to fixed-point values. This greatly reduces execution time thanks to the lower computational cost of fixed-point operations, but it also reduces the accuracy of neural networks. Depending on the structure of a neural network, its size, and its overall robustness to weight quantization, reducing the precision of network weights will have a larger or smaller impact on the results. In our expressive technique classifier, weight quantization reduced greatly the accuracy of the results (i.e., 10-20% depending on different quantization techniques) so we avoided using it in the final models.

3.3. Audio processing deadlines and real-time-safe programming rules

Section 2.3 described briefly how audio is delivered in buffers to any processing software, including rt-MIR systems, leaving a set "time-budget" for the completion of signal processing and analysis computations. In this context, rt-MIR systems that need to execute their entire analysis pipeline at each buffer-read operation will have a strict constraint on the number of operations that can be performed to produce their results. In these cases, the size of the audio buffer can be increased, but it will slow down the entire analysis. Furthermore, many rt-MIR algorithms can be subdivided into multiple stages of analysis, where some can potentially run at a different rate of execution without affecting the time-budget available. For example, the expressive technique recognition pipeline presented in Section 4.3 is composed of an onset detector, a set of feature extractors, and a deep learning classifier (see Fig. 3). The onset detector must run in the real-time thread of execution and will benefit from running at a high rate (i.e., small audio buffer size), which reduces the latency with which onsets are detected. On the contrary, features need to be extracted only when an onset is detected, and the same holds for the execution of the rather computationally expensive deep learning classifier. Moreover, the requirements of our task state that, as a reasonable assumption, there is no need to classify notes that are less than 20 ms apart, so both the feature extraction operation and the execution of the classifiers are rather low-rate tasks. The onset detector was executed in the real-time thread of execution with a buffer size of 64 samples at 48 kHz (i.e., onset detection is executed every 1.33 ms), while both the feature extraction and classification stages can exceed the time budget of the audio processing routine (i.e., 1.33 ms), and their execution can be moved to a separate high-priority thread.

Nevertheless, this divide-et-impera approach can not be applied to end-to-end neural networks. In fact, an underlying trend in deep learning is to devise end-to-end neural networks, which can operate directly on raw data and produce their results without any additional algorithm. This approach leads neural networks to learn "internal" feature extractors that can surpass hand-crafted feature extractors in terms of the quality of the descriptors, but this approach can greatly increase the computational complexity of the analysis pipeline. Additionally, end-to-end neural networks can hardly be subdivided into different stages.

Apart from reducing the execution time of the code devised

by the developer of a rt-MIR system, great care must be devoted to using external libraries that only perform real-time-safe operations during the execution of the analysis of the signal (see Section 2.3). For example, our expressive technique recognition system uses the real-time onset detector from the Aubio library [25], and our feature extractors were C++ ports of the TimbreID library tools [26]. Moreover, there are various deep learning execution libraries that can run inference of neural networks quickly and without breaking real-time-safe programming rules (i.e., TensorFlow Lite, ONNX Runtime, and RTNeural [27]).

3.4. Embedded hardware and software limitations

As mentioned in Section 3.4, modern deep learning models can be computationally expensive to train and execute. On powerful computers, training, and execution of neural networks are accelerated thanks to GPUs, which are specialized hardware for high-throughput parallel operations. However, very few embedded computers are provided with specialized hardware that can support deep learning tasks. Among these, the Nvidia Jetson family of embedded computers is specifically designed to accelerate deep learning applications with the use of GPUs. More recently, many different types of processing units for low-power deep learning inference have been developed, such as Google's Coral TPU and Intel's Visual Processing Units (VPUs). Some of these come in the form of an embedded computing board (e.g., the Coral Dev Board[2]), while others are provided as external devices that communicate with the CPU of embedded computers through a USB connection (e.g., Coral USB Accelerator[3], Neural Compute Stick[4]).

Additionally, the more flexible design of FPGAs has become increasingly more interesting for deep learning inference. This is to be attributed to the development of new tools to convert neural networks to FPGA compatible code. Vandendriessche et al. [9] recently compared the performance of multiple embedded alternatives for environmental sound recognition, which include TPUs and FPGAs. The authors focused on Convolutional Neural Networks (CNNs), which is a type of neural network that require a large number of parallelizable computations. FPGAs have also been explored for ultra-low latency DSP. In particular, Risset et al. presented preliminary results on the development of a tool that can compile Faust code for FPGAs [11], while very recently Wegener et al. described a method to interface Pure Data with a FPGA for low latency physical modeling synthesis [28].

However, single-board computers have become more powerful in recent years, resulting in the possibility of executing rather small neural networks, such as compact Feed-Forward Neural Networks (FFNNs), with only a CPU. Moreover, each of the aforementioned acceleration hardware solutions introduces a different overhead because of the communication with the CPU, which may become an issue for very low-latency real-time systems

Apart from the aforementioned hardware alternatives, embedded rt-MIR systems can greatly benefit from using optimized real-time audio platforms [8]. Some of the most advanced alternatives are the Bela platform [29] and the Elk audio operating system (Elk Audio OS) [30]. Both of these solutions are based on the Xenomai Cobalt real-time kernel, which allows reaching very low scheduling latency ad round-trip audio latency. Moreover, Vignati et al. [31] recently compared the performance of the Xenomai kernel with the more accessible PREEMPT_RT patch for the Linux kernel. The comparison showed the superior performance of Xenomai-based solutions, but PREEMPT_RT proved to be considerably easier to implement while still delivering a good performance. Overall, these low-level software solutions can allow embedded rt-MIR systems to exploit more CPU performance and reduce latency when compared to the standard Linux kernel. Our expressive technique classifier is deployed to a Rasberry PI computer that runs Elk Audio OS and neural inference is executed on the CPU because of the aforementioned considerations.

[2]https://coral.ai/products/dev-board/

[3]https://coral.ai/products/accelerator

[4]https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html

4. EXPRESSIVE TECHNIQUE CLASSIFIER

To demonstrate some of the potential solutions described in the previous section, we now present a real-time embedded classification system that can recognize the expressive playing technique used by an acoustic guitar player. This system is limited to a monophonic setting, where the musician plays one note at a time, similarly to a guitar solo setting. Furthermore, we performed the detection on the acoustic guitar, which allows for a great range of techniques such as percussive techniques (i.e., drum-like sounds produced by hitting the body of the instrument). Real-time detection of the type of percussive hits can be used to trigger drum samples or control percussive synthesis algorithms. The classifier was deployed on a Raspberry PI 4 embedded computer with Elk Audio OS. The code of the expressive guitar technique classifier is available in an online repository[5].

Figure 1: *Hardware setup of the expressive technique classifier. The audio signal from the guitar pickup is fed to a Raspberry PI through the Elk PI Hat, which contains high definition audio converters. The result of the classification is communicated through a simple sine wave tune with different frequencies, which is monitored with a pair of headphones.*

The technique classifier was trained on three tasks of increasing complexity, which are described in Section 4.1. Section 4.2 describes the dataset used to train the classifier, while Section 4.3 illustrates the detail of the classification pipeline. Finally, Section 4.4 presents the results of the system in terms of accuracy and latency.

4.1. Classification tasks

This study was divided into three classification tasks of increasing level of complexity:

[5]https://github.com/domenicostefani/cpp-timbreID/

- *Task A* [binary]: The first task is a binary classification problem, where the output of the neural network must indicate whether a note was produced with a *pitched* or *percussive* technique;
- *Task B* [percussive+]: The second task involves multiclass classification of four individual percussive techniques and a single aggregate class for every pitched technique (See next section);
- *Task C* [full]: The third task is the full classification problem, where twelve expressive playing techniques are considered as individual classes. This problem is considerably more complex than Task A and B.

4.2. Dataset

This study required a dataset containing high-quality recordings from multiple acoustic guitars with timbre annotations for each note played. Moreover, the recordings needed to be monophonic and to contain a number of relevant playing techniques. Finally, as the final system must work whitout external microphones, the dataset had to be recorded through pickups embedded into the guitar. Given the scarcity of free datasets of guitar recordings complying with our specific requirements, we proceeded to define and record a new dataset. With the help of professional guitarists we compiled the following list of twelve acoustic guitar techniques, which includes both percussive and pitched techniques:

1. *"Kick" percussive technique*: producing a sound that resembles a kick drum by hitting the lower right part of the top of the guitar body;
2. *"Snare-A" percussive technique*: producing a sound by hitting the lower right side of the guitar body;
3. *"Tom" percussive technique*: producing a sound by hitting the area of the guitar body near the top of the end of the fretboard, using the thumb;
4. *"Snare-B" percussive technique*: producing a sound by hitting the muted strings over the end of the fretboard;
5. *Bending technique*: pulling the strings, raising the pitch (half-tone interval);
6. *Hammer-on technique*: sharply bringing a finger down onto the fingerboard, creating a legato sound (half-tone interval);
7. *Natural Harmonics*: plucking the strings and stopping them to suppress the dominant note frequency and let harmonic overtones ring;
8. *Palm Mute*: partially muting the strings with the palm of the picking hand, resulting in a muffled sound.
9. *"Pick Near Bridge"*: plucking the string near to the guitar bridge, producing sounds with great high-frequency content;
10. *"Pick Over the Soundhole"*: plucking the string over the soundhole, producing sounds with lower treble content and greater intensity;
11. *Staccato*: playing short notes;
12. *Vibrato*: Moving the fretting-finger to warp the pitch and tone of the sound.

The selection of percussive techniques was inspired by the interview study on percussive fingerstyle conducted by Martelloni et al. [32]. The areas of the guitar body used for each of these techniques are shown in Fig. 2.

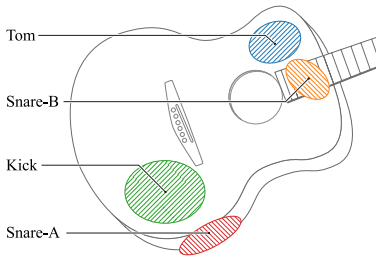

Figure 2: *Percussive techniques of choice, with the relative guitar areas used.*

The dataset was recorded with five experienced guitarists and five different guitars, using the internal pickups of each guitar. The percussive techniques were recorded for each guitarist with three intensities (piano, mezzoforte, forte), with 10 repetitions for each sound. For all the pitched techniques, individual notes were played across a selected ranges of keys[6] for each string. For each key on each string, the corresponding note was played three times for each of the aforementioned intensity levels. While the dataset has yet to be labeled at onset level, which will be done in the near future, the onset detector used for our implementation indicated that the total of our recordings for the dataset contains about 35,035 notes.

4.3. System Architecture

This section describes the design of the classification pipeline and its practical implementation for embedded real-time expressive guitar technique recognition. The target embedded device for our classifier is the Raspberry PI 4 (4 GB RAM version) with Elk Audio OS, the choice of which was based on the considerations drawn in Section 3.4. In order to comply with the constraints imposed by the limited computational power of the target embedded platform, we choose to use a multi-step classification pipeline (see Section 3.3), composed of an onset detector, a set of feature extractors, and a deep learning classifier (See Fig. 3).

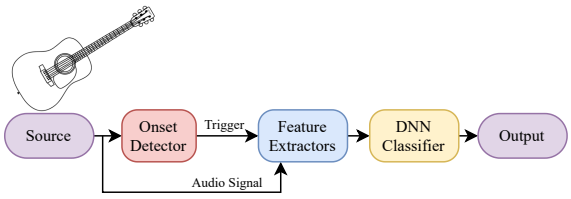

Figure 3: *Classification pipeline.*

This allows us to separate the steps that need to be executed with a high refresh rate (i.e., onset detection), from those that only need to be executed with a lower frequency (i.e., feature extraction, classification). Moreover, classification can be performed on a thread separate from the real-time execution. As a result, the latency introduced by the software system can be described by the sum of the latency of each stage of the pipeline (See Fig. 4). Each

[6]Natural harmonics were recorded only for frets 5, 7 and 12, while the remaining pitched techniques were played from the open string up to a fret between the 15th and the 20th, depending on the physical limitations of each guitar (e.g., cutaway, string gauge, guitar scale)

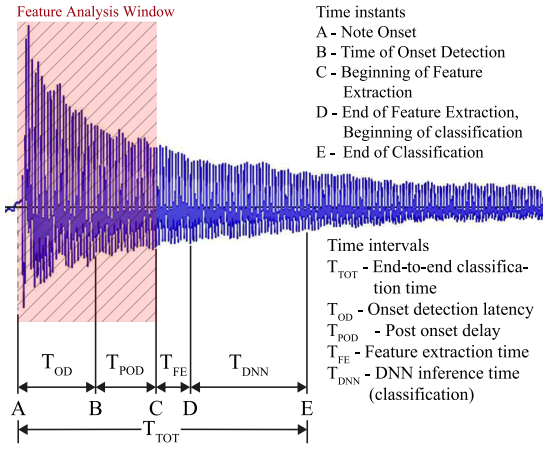

Figure 4: *Graphical representation of a guitar sound in the audio signal and all the components of the total latency between a note onset and its classification.*

step of the pipeline is now described in detail.

Onset Detection is the task of recognizing the beginning of individual sounds in an audio signal (i.e., onsets). Contrary to the technique classifier at the end of the pipeline, the onset detector has to analyze the audio signal at a fast rate. This imposes a constraint on the types of detectors that can be used, ruling out slow offline solutions like most deep learning approaches. We found *aubioonset* [25] to be reliable and able to detect onsets with a small latency interval.

Feature Extraction: While it is possible to devise a neural network that operates directly on the raw audio signal, its high dimensionality can be counterproductive for discriminative tasks [33]. For these tasks, an alternative solution is to exploit transformations that better describe the relevant signal properties of interest. On the full dataset, the following feature extractors of the TimbreID library were used: Attack time, Bark Spectrum Brightness, Bark Spectrum (50 values), Bark Frequency Cepstral Coefficient (BFCC) (50 values), Cepstrum, Mel Frequency Cepstral Coefficient (MFCC), and Peak sample. As introduced in Section 3.1, the feature extraction was delayed as much as possible while fitting into the target latency, resulting in a feature analysis window of 1,024 samples, which corresponds to 21.33 ms of audio at 48 kHz. To do so, the computation of features was delayed by a brief time interval after each onset detection (i.e., Post-onset delay), to align the beginning of the window with the note onset. For each classification task, a different feature subset was selected through a process of trial and error.

Classification: The classification of the expressive playing technique was performed through neural network models loaded in TensorFlow Lite. Each model was developed through a process of trial and error, by testing multiple combinations of input features, layer number and size, dropout probabilities, and optimization parameters. The Stochastic Gradient Descent optimizer was found to work best for all the learning tasks and was used with a learning rate of $1e{-}3$ and momentum values between 0.7 and 0.9. The loss function used was Sparse Categorical Cross-entropy, as it is appropriate for multiclass classification problems. The execution time of models of different sizes was measured on the target platform, and the size of the final models was tuned to fit the target latency (see Section 3.2). Each neural network was trained for a number of epochs between 1,000 and 2,000, on a 75% stratified random split of the original dataset. The following three neural networks were used for Task A, B, and C:

- *Network A* [binary]: Network A uses 110 input features (50 BFCC + first 60 Real Cepstrum features) and it is composed of four dense hidden layers (fully connected) with 500 neurons each. The final layer has 2 neurons. Each of the hidden layers used the LeakyRelu activation function, end they were interleaved by dropout layers for regularization, as suggested by Sigtia et al; [34].
- *Network B* [percussive+]: The model used for task B was similar to that of Task A, differing in the number of output neurons (i.e., five outputs for Network B). The first four outputs represent the prediction of one of the four percussive techniques, while the fifth corresponds to any pitched sound;
- *Network C* [Full]: Similarly, the same network of Task A and B was used for the full classification task, except for using 513 input features (i.e., all the Cepstrum coefficients) and having 12 outputs, one for each expressive technique.

4.4. Results and Discussion

The success rate of the classifier was measured in terms of its accuracy, precision, recall and F1-score. These were measured on a stratified random split of 25% of the entire dataset, while each neural network was trained only on the remaining 75% split. The best results obtained through a process of trial and error for Task A, B and C are presented respectively in Table 1, 2 and 3.

Table 1: *Summary of the results of Task A.*

Class	Precision	Recall	F1-score
Percussive	95.6%	92.6%	94.1%
Pitched	99.5%	99.7%	99.6%
Macro avg.	97.5%	96.1%	96.8%
Accuracy			**99.2%**

Table 2: *Summary of the results of Task B.*

Class	Precision	Recall	F1-score
Kick	91.4%	95.5%	93.4%
Snare-A	89.3%	95.0%	92.1%
Tom	92.8%	91.8%	92.3%
Snare-B	90.9%	96.8%	93.7%
Pitched	99.7%	99.4%	99.5%
Macro avg.	92.8%	95.7%	94.2%
Accuracy			**99.1%**

Table 3: *Summary of the results of Task C.*

Class	Precision	Recall	F1-score
Kick	82.1%	76.7%	79.3%
Snare-A	78.3%	66.7%	72.0%
Tom	64.7%	39.3%	48.9%
Snare-B	78.3%	60.0%	67.9%
Bending	55.7%	42.8%	48.4%
Hammer-on	48.0%	48.8%	48.4%
Nat.Harmonics	69.7%	50.9%	58.8%
Palm Mute	61.1%	82.0%	70.0%
Bridge-Pick	67.2%	59.8%	63.3%
Soundhole-Pick	49.4%	52.6%	51.0%
Staccato	60.4%	71.2%	65.3%
Vibrato	48.5%	41.6%	44.8%
Macro avg.	63.6%	57.7%	59.8%
Accuracy			**56.5%**

These results show how both Task A and Task B can be tackled effectively with a rather simple FFNN method, using a naive trial and error process to establish the input features to use and the parameters of the classifier. In particular, the classification accuracy of Task B (99.1%) surpassed our previous results obtained with smaller feature vectors on the simpler task of only percussive technique recognition (i.e., 92.5%). On the other hand, Task C is considerably more complex, and we were not able to reach a meaningful accuracy score. Preliminary results seem to indicate that reducing the variance of the onset detection latency will lead to more precise alignment of the feature extraction window, which in turns should improve classification accuracy. Moreover, feature selection can be used to obtain better feature vectors, and more regularization layers in the deep classifier (e.g., Batch Normalization) may allow the neural network to be trained more effectively.

4.4.1. Latency

Measuring the exact end-to-end latency of the classification system would require feeding the embedded computer with a prerecorded guitar signal, having the classifier produce a clear signal mark in output when classification is completed, and hand-labeling each onset and mark in the signal. This would allow for a precise measurement of the delay introduced by the software between onsets and classification results.

For the sake of simplicity, an approximate measuring setup was used for this demonstrative case. Each delay was measured on Task B. The delay between an onset and its detection (Onset detection latency) was measured separately, on 211 individual notes (about 100 percussive sounds and 100 pitched ones), and it averaged at 19.00 ms. It is to be noted that the latency of detection depends on changes in the inner buffer of the onset detector, and it is different from the execution time of the detector itself, which follows each audio buffer read operation and is considerably shorter than the 1.33 ms time-budget of our real-time system (64 samples at 48 kHz).

Instead, all the delays apart from onset detection latency could be measured inside the classification software in real-time, with a high precision timer. These were measured by playing 200 notes (100 percussive and 100 pitched) with the guitar connected to the system, and saving each timer result in a log file. Such delays were composed of the post-onset delay, the feature computation time and finally the inference time of the classifier (see Fig. 4), which averaged respectively at 7.77 ms, 0.78 ms, and 3.15 ms. These delays add up at 11.70 ms. As an indicative measure, the sum the onset detection latency and the remaining delays totals at 30.7 ms. However, the next iteration of the expressive guitar technique classifier will include a more precise latency test setup.

Most importantly, the system was complemented with a simple sine-wave oscillator that played a different tune depending on the classification result, as soon as it was produced. The system was then connected to the guitar and played in real-time, and the delay between the act of the guitarist and the synthetic tune was inaudible. This is a subjective but promising result, as it suggests that a smart guitar could use a similar system to trigger different synthetic sounds depending on the technique used, without the player perceiving the recognition delay.

5. CONCLUSION AND FUTURE WORKS

In this paper, we discussed the challenges of developing embedded real-time Music Information Retrieval systems for acoustic signals. These included the availability of only past and current input information, the process of tuning the tradeoff between system accuracy and latency, real-time audio deadlines, real-time-safe programming rules, and the limitations of embedded hardware and low-level software. Furthermore, we presented potential solutions to these issues, and we demonstrated the validity of some of these with the embedded implementation of a real-time expressive guitar technique classifier.

The demonstrative implementation reached 99.2% accuracy in distinguishing pitched and percussive techniques and a 99.1% average accuracy in distinguishing four distinct percussive techniques and a fifth class for pitched sounds. On the contrary, the full classification task of distinguishing twelve different techniques was more complex, and the proposed approach could not reach satisfactory accuracy scores. The classification pipeline was successfully deployed on an embedded computer, and the classification results were produced with an average latency of approximately 30.7 ms from the note onset, which was hardly perceptible. The solutions demonstrated by our experiment are the possibility of dividing the classification pipeline in several steps that execute at different frequencies, the tuning of the system delays, the effective use of a real-time embedded OS to obtain the best performance on a resource constrained device and the use of real-time-safe code. Furthermore, the results of this study and the challenges discussed can also apply to more general real-time audio contexts, such as in acoustic sensor networks, where classification may need to be performed with very low latency.

However, the demonstrative experiment is limited in several ways. First of all, the complex "full" classification problem will require an improvement of the entire recognition pipeline to achieve satisfying results. Moreover, the classification latency can be further reduced, to enable chaining more complex synthesis algorithms and effects after the classification. Additionally, the proposed classifier works with the guitarist playing only one note at at time, such as in a guitar solo setting. Finally, an objective and repeatable measurement of the end-to-end latency will require the design of a custom test setup.

Future studies will devote more attention to the refinement of the entire classifier, to reduce latency and improve the classification accuracy. Moreover, we will attempt to use the separate signals from an hexaphonic pickup to enable polyphonic technique recognition. Finally, a future development of the project should investigate the potential of Tensor Processing Units and other modern acceleration hardware in the context of embedded real-time Music Information Retrieval.

6. REFERENCES

[1] S. Böck and M. Schedl, "Enhanced beat tracking with context-aware neural networks," in *Proc. 14th Int. Conf. on Digital Audio Effects (DAFx-11)*, 2011, pp. 135–139.

[2] J. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. Sandler, "A tutorial on onset detection in music signals," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, 2005.

[3] S. Sigtia, E. Benetos, and S. Dixon, "An end-to-end neural network for polyphonic piano music transcription," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, pp. 927–939, 2016.

[4] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[5] E. Berdahl, "How to make embedded acoustic instruments," in *Proc. Int. Conf. on New Interfaces for Musical Expression (NIME)*, London, United Kingdom, June 2014, pp. 140–143, Goldsmiths, University of London.

[6] G. Moro, A. Bin, R. H. Jack, C. Heinrichs, A. P. McPherson, et al., "Making high-performance embedded instruments with bela and pure data," in *Proc. Int. Conf. on Live Interfaces (ICLI)*. 2016, University of Sussex.

[7] L. Turchet, "Smart Musical Instruments: vision, design principles, and future directions," *IEEE Access*, vol. 7, pp. 8944–8963, 2019.

[8] A. McPherson, R. Jack, and G. Moro, "Action-sound latency: Are our tools fast enough?," in *Proc. Int. Conference on New Interfaces for Musical Expression (NIME)*, Brisbane, Australia, 2016, pp. 20–25, Queensland Conservatorium Griffith University.

[9] J. Vandendriessche, N. Wouters, B. da Silva, M. Lamrini, M. Y. Chkouri, and A. Touhafi, "Environmental sound recognition on embedded systems: From fpgas to tpus," *Electronics*, vol. 10, no. 21, 2021.

[10] R. Michon, Y. Orlarey, S. Letz, and D. Fober, "Real Time Audio Digital Signal Processing With Faust and the Teensy," in *Sound and Music Computing Conference (SMC-19)*, Malaga, Spain, May 2019.

[11] T. Risset, R. Michon, Y. Orlarey, S. Letz, G. Müller, and A. Gbadamosi, "Faust2FPGA for Ultra-Low Audio Latency: Preliminary work in the Syfala project," in *Second Int. Faust Conference*, Paris, France, Dec. 2020, pp. 1–9.

[12] L. Reboursière, O. Lähdeoja, T. Drugman, S. Dupont, C. Picard-Limpens, and N. Riche, "Left and right-hand guitar playing techniques detection," in *Proc. Int. Conf. on New Interfaces for Musical Expression (NIME)*, Ann Arbor, Michigan, 2012.

[13] L. Reboursière, O. Lähdeoja, R. Chesini Bose, T. Drugman, S. Dupont, C. Picard-Limpens, and N. Riche, "Guitar as controller," *Numediart Quartely Progress Scientific Report*, vol. 4(3), 2011.

[14] A. Martelloni, A. McPherson, and M. Barthet, "Guitar augmentation for percussive fingerstyle: Combining self-reflexive practice and user-centred design," in *Proc. Int. Conf. on New Interfaces for Musical Expression (NIME)*, Shanghai, China, 2021.

[15] B. Bhattarai and J. Lee, "Automatic music mood detection using transfer learning and multilayer perceptron," *The International Journal of Fuzzy Logic and Intelligent Systems*, vol. 19, no. 2, pp. 88–96, Jun 2019.

[16] Y. Yu, S. Luo, S. Liu, H. Qiao, Y. Liu, and L. Feng, "Deep attention based music genre classification," *Neurocomputing*, vol. 372, pp. 84–91, 1 2020.

[17] F. Eyben, S. Böck, B. Schuller, and A. Graves, "Universal onset detection with bidirectional long-short term memory neural networks," in *Proc. 11th Intern. Soc. for Music Information Retrieval Conference, ISMIR*, 2010, pp. 589–594.

[18] D. Stowell and M. Plumbley, "Adaptive whitening for improved real-time audio onset detection," in *Proc. 2007 Int Computer Music Conf. ICMC 2007*. University of Surrey, 2007, pp. 312–319.

[19] S. Böck, F. Krebs, and M. Schedl, "Evaluating the Online Capabilities of Onset Detection Methods.," in *Proc. of the 13th Int. Society for Music Information Retrieval Conf.*, Porto, Portugal, Oct. 2012, pp. 49–54, ISMIR.

[20] B. C. J. Moore, *An introduction to the psychology of hearing*, Brill, 2012.

[21] O. Kramer, "K-nearest neighbors," in *Dimensionality reduction with unsupervised nearest neighbors*. 2013, vol. 51, pp. 13–23, Springer.

[22] R. Bencina, "Interfacing real-time audio and file i/o," in *Proc. of the Australasian Computer Music Conference (ACMC)*, 2014, pp. 21–28.

[23] S. Böck, A. Arzt, F. Krebs, and M. Schedl, "Online real-time onset detection with recurrent neural networks," in *Proc. 15th Int. Conf. on Digital Audio Effects (DAFx-12), York, UK*, 2012.

[24] S. Sigtia, E. Benetos, and S. Dixon, "An end-to-end neural network for polyphonic piano music transcription," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 5, pp. 927–939, 2016.

[25] P. M. Brossier, *Automatic Annotation of Musical Audio for Interactive Applications*, Ph.D. thesis, Queen Mary University of London, 2006.

[26] W. Brent, "A timbre analysis and classification toolkit for pure data," in *Proc. 2010 Int. Computer Music Conf., ICMC 2010, New York, USA, 2010*. 2010, Michigan Publishing.

[27] J. Chowdhury, "Rtneural: Fast neural inferencing for real-time systems," *arXiv preprint arXiv:2106.03037*, 2021.

[28] C. Wegener, S. Stang, and M. Neupert, "Fpga-accelerated real-time audio in pure data," in *Proc. Int. Conf. in Sound and Music Computing, SMC-22*. June 2022, Zenodo.

[29] A. McPherson and V. Zappi, "An environment for submillisecond-latency audio and sensor processing on beaglebone black," in *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015.

[30] L. Turchet and C. Fischione, "Elk Audio OS: an open source operating system for the Internet of Musical Things," *ACM Transactions on the Internet of Things*, vol. 2, no. 2, pp. 1–18, 2021.

[31] L. Vignati, S. Zambon, and L. Turchet, "A comparison of real-time linux-based architectures for embedded musical applications," *Journal of the Audio Engineering Society*, vol. 70, no. 1/2, pp. 83–93, 2022.

[32] A. Martelloni, A. McPherson, and M. Barthet, "Percussive Fingerstyle Guitar through the Lens of NIME: an Interview Study," in *Proc. Int. Conf. on New Interfaces for Musical Expression (NIME)*, 2020, pp. 525–531.

[33] J. Turian, J. Shier, H. R. Khan, B. Raj, B. W. Schuller, C. J. Steinmetz, C. Malloy, G. Tzanetakis, G. Velarde, K. McNally, et al., "Hear 2021: Holistic evaluation of audio representations," *arXiv preprint arXiv:2203.03022*, 2022.

[34] S. Sigtia and S. Dixon, "Improved music feature learning with deep neural networks," *ICASSP, IEEE Int. Conf. on Acoustics, Speech and Signal Processing - Proceedings*, pp. 6959–6963, 2014.