Towards Real-Time Detection of Symbolic Musical Patterns: Probabilistic vs. Deterministic Methods

Nishal Silva
Independent researcher
Colombo, Sri Lanka
nishals@ieee.org

Carlo Fischione
KTH Royal Institute of Technology
Stockholm, Sweden
carlofi@kth.se

Luca Turchet
University of Trento
Trento, Italy
luca.turchet@unitn.it

***Abstract*—The computational detection of musical patterns is widely studied in the field of Music Information Retrieval and has numerous applications. However, pattern detection in real-time has not yet received adequate attention. The real-time detection is important in several application domains, especially in the field of the Internet of Musical Things. This study considers a single musical instrument and investigates the detection in real-time of patterns of a monophonic music stream. We present a representation mechanism to denote musical notes as a single column matrix, whose content corresponds to three key attributes of each musical note - pitch, amplitude and duration. The note attributes are obtained from a symbolic MIDI representation. Based on such representation, we compare the most prominent candidate methods based on neural networks and one deterministic method. Numerical results show the accuracy of each method, and allow us to characterize the trade-offs among those methods.**

I. Introduction

Music is a systematic organization of musical notes of varying duration and intervals of silence. The presence of repetitive patterns is one of the most significant aspects of music. For instance, considering monophonic signals, a musical pattern can be defined as an ordered sequence of notes and pauses. Research has shown that human listeners usually associate a piece of music with its underlying patterns [1], [2].

The computational detection of repetitive patterns in music has been widely studied in the field of Music Information Retrieval [3]. There are numerous applications of musical pattern detection, including computational musicology, beat tracking, audio fingerprinting, or audio mixing. Most of existing studies have focused, however, on offline contexts, i.e., by analizing recordings, whereas little attention has been devoted to the real-time scenario, i.e., when the analysis needs to be performed at the moment in which the musical pattern is generated by a musical instrument.

Real-time pattern detection from the output of a musical instrument is an essential aspect in those Internet of Musical Things (IoMusT) applications [4] based on smart musical instruments [5]. In this family of musical instruments, the embedded intelligence can be exploited to extract, in real-time, patterns from the player's output and repurpose them into controls for connected devices. These devices include smoke machines, stage lights, visuals or the audio-visual output of smartphones of audience members in participatory live music contexts. For instance, a smart cajón player [6] during a live concert could play a pre-determined rhythmic pattern and in response to it a control could be automatically triggered to modify the color of stage lights. To date, scarce research has been conducted to address this kind of scenarios envisioned in IoMusT research [4].

In this paper, we develop and evaluate the most prominent methods to detect, in real-time, patterns in monophonic musical signals related to a single instrument. In particular, we compare the performance and complexity of both probabilistic and deterministic systems. Our study assumes that a smart musical instrument converts the incoming audio signal into MIDI notes (as performed, for instance, by the smart cajón reported in [6]). Therefore, we have not considered the actual audio signal in the form of digitized audio waves, but its symbolic representation in the MIDI format. In more detail, we aim at detecting, with minimum latency, a set of pre-determined patterns as soon as they appear in a running monophonic MIDI sequence, exactly how it would happen in a real live music context. With this study we aim to contribute to the development of technological solutions that enable the creation of IoMusT applications by pre-programming smart musical instruments with patterns chosen by the musician, producer or composer.

The remainder of this paper is organized as follows: Section II introduces several other existing studies that have been conducted on the identification of musical patterns. Section III elaborates on the musical notation format introduced in this study, while Section IV explains our definition of musical patterns and their tolerances. Sections V and VI provide detailed explanations on the methods that we have used. Section VII provides an overview of the datset and the results. Section VIII presents the results of the study, and discusses possible extensions and section IX Presents our conclusions.

II. Related Works

The detection of patterns in a recorded musical signal has been widely studied and several implementations are available. Although most of the existing studies aim to detect patterns in recorded music, some methods could potentially be modified to work in real-time. This section overviews the most relevant literature on the topic of pattern detection in music.

It is possible to categorize these methods into different categories based on the representation of the audio stream. Some of such categories that we have analyzed in this section are:

- String based analysis methods

- Tree based analysis methods
- Spectrum based analysis methods
- Correlative matrix analysis based methods
- Geometrical analysis methods

String based analysis methods represent the audio stream in the form of a string. The work in [7] presents a technique to extract patterns from a given melodic line. This method is iterative and not intended for a real-time application. An iterative program scans the beginnings and the ends of patterns. Discovered patterns are deemed significant by analyzing their length and frequency of occurrence, and non overlapping, immediately repeating patterns are considered special cases.

The study in [8] proposes a method to identify specific character sequences from larger texts and explores several deep learning methods. The authors present algorithms based on convolutional networks, recurrent networks and a hybrid both. The predictions for several text patterns are evaluated. The study concludes that deep learning methods are able to achieve a high accuracy precision in sequence pattern recognition.

A method to classify musical patterns extracted from raw audio is presented in [9]. The study presents an algorithm for pattern extraction and their conversion to sequences of musical intervals, which are input to a set of variable-duration Hidden Markov Models (HMM). Each HMM has been trained to detect patterns belonging to its corresponding class. This classification method has been used in the context of Greek traditional music, specifically on monophonic musical patterns generated by the Greek traditional clarinet. The authors of [9] report difficulties of Greek traditional music when compared to other Western music, which makes pattern recognition more difficult. The research showed the improvements of the proposed method over methods based on conventional HMMs.

A similarity measure for multivariate time series datasets, which are common in various multimedia, medical and financial applications, is presented in [10]. This method is based on Principal Component Analysis. Although intended to measure similarity between time series, this method has the possibility to be adapted for pattern detection through the use of a windowing function. The author used the principal components and eigenvalues to compare the similarity between multivariate time series matrices. The results obtained by using several datasets showed that the method performed better compared with traditional approaches (such as Euclidean Distance, Dynamic Time Warping, Weighted Sum SVD and PCA similarity factor).

Tree based analysis methods represent the audio stream as a tree structure and use properties of this structure to identify patterns. The authors of [2] present such a method, which aims to use the repeating patterns to speed up music retrieval, based on the fact that repeating patterns are key melodies and are easier to familiarize with people. The involved algorithm use the musical signal to construct a suffix tree, leveraging algorithms presented in [11]. The authors identify repeating patterns by scanning the nodes of the constructed tree.

Some methods use a spectral representation of the audio to identify patterns. The work in [12] aims to find repeating patterns in music to perform the task of separating the repeating background from the non-repeating foreground in a musical mixture. The authors propose an algorithm able to identify periodically repeating segments in the audio through an autocorrelation of the Short Time Fourier Transform spectrum and the power spectrum. The authors presents the results on data sets of 1000 song clips and 14 full-track real-world songs. The algorithm is also proposed to be used as a preprocessing stage for pitch detection algorithms, to improve melody extraction.

The authors of [13] introduces a data structure called the "correlative matrix", which is used by several subsequencet studies as well. The correlative matrix is used to store the information extracted from the music. The correlative matrix is then utilized to determine repeating patterns and their locations. The authors of the study described in [14] present a modified version of the correlative matrix introduced in [13] to increases the efficiency of the algorithm.

Some methods use the geometrical nature of music as a basis for its representation. The authors of [15] present a method to identify repeated patterns in music by representing the music as a multidimensional dataset, of which two dimensions are plotted. These scatter plots are utilized to find repeating patterns present in music. The work reported in [16] presents a method to extract repeating drum loops from polyphonic music. The spectrogram is used to cluster the onset signal into known drum types, which are used to determine repeating patterns. In a different vein, the mechanisms behind the popular music identification program Shazam [17] is presented in [18]. This study elaborates on the creation of fingerprinting hashes, which are then used to match an audio segment with a database.

Another geometrical analysis method representing the musical melodies as curves is presented in [19]. The minimum area between the curves is utilized to measure the degree of match. This method can account for pitch and time variations in similar curves. The authors of [20] also present a similar geometrical approach, where each note is represented as a horizontal line segment in a time vs. pitch plane. These line segments are translated in the plane to find matches. Such a work further categorizes the matched patterns based on the pitch change.

Notably, the majority of the studies overviewed above are not directly targeting the real-time domain.

III. Representation of Musical Notes

In this study, we propose to use a mechanism to represent musical notes using three of their primary attributes. Each note is represented as a column matrix with three elements, each of which corresponds to the following note attributes:

- Pitch: the perceived height of the musical note, which is related to the actual frequency of it;
- Note duration: the time interval between the note onset and its end;
- Amplitude: the highest peak of the note through its duration.

A sample representation of a musical note can be presented as follows:

$$C_i = \begin{bmatrix} P_i \\ A_i \\ D_i \end{bmatrix}. \quad (1)$$

In the above sample note C_i, the pitch is denoted by P_i, the amplitude is denoted by A_i, and the note duration is denoted by D_i. Similarly, a sequence of musical notes S can be represented as follows:

$$S = \begin{bmatrix} P_i \\ A_i \\ D_i \end{bmatrix}, \begin{bmatrix} P_{i+1} \\ A_{i+1} \\ D_{i+1} \end{bmatrix}, \begin{bmatrix} P_{i+2} \\ A_{i+2} \\ D_{i+2} \end{bmatrix}, \dots$$

Pitch. The pitch of a note depends primarily on frequency at which the sound wave associated with it resonates, and on loudness and spectrum. Classical music and most Western music has defined the standard pitch which is based on the 12 tone equal temperament (12-TET) tuning system. This tuning system divides an octave into 12 equal parts, which are then mapped to the 12 notes in Western music [21]. The standard 12-TET tuning system defines the A_4 note (the A note above middle C in a standard piano) - as 440Hz. In this study, we use the standard 12-TET pitch, which is converted to MIDI notes for simpler visualization.

We use the MIDI notation to represent the pitch. The MIDI numbers are defined based on the MIDI keyboard, which has 128 keys. Each note is mapped to an integer to represent its frequency. The MIDI notes span from *0* ($C_{-1} \approx 8.18Hz$), to *127* ($G_9 = 12.5kHz$). The MIDI note number for $A_4 = 440Hz$ is *69*.

Amplitude. The amplitude is a measurement of the loudness of the musical note. A key characteristic of a musical sequence is the relative loudness and softness between notes or phrases of music. In Western music, the variation between the relative loudness is defined as *Dynamics*. There are two basic dynamic indicators in written music which are denoted by $\boldsymbol{p}$ (piano) - meaning *soft*, and $\boldsymbol{f}$ (forte) - meaning *loud* or *hard*. It is common for the usage of up to three $\boldsymbol{p}$'s of $\boldsymbol{f}$'s to covey varying degrees of loudness. Some of the most common dynamic markers are [22]:

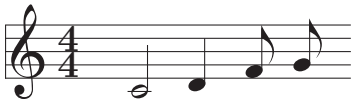
- $\boldsymbol{ppp}$ - *pianissimissimo*, meaning very very soft.
- $\boldsymbol{pp}$ - *pianissimo*, meaning very soft
- $\boldsymbol{p}$ - *piano*, meaning soft
- $\boldsymbol{mp}$ - *mezzo-piano*, meaning moderately soft
- $\boldsymbol{mf}$ - *mezzo-forte*, meaning moderately loud
- $\boldsymbol{f}$ - *forte*, meaning loud
- $\boldsymbol{ff}$ - *fortissimo*, meaning very loud
- $\boldsymbol{fff}$ - *fortissimissimo*, meaning very very loud

The amplitude is obtained using the *velocity*, which is defined by the MIDI standard. This measurement, like for the pitch, is also represented by an integer in the range of 0-127. The popular production software Logic Pro X maps the MIDI velocity to musical dynamics markers as shown in Table I [23].

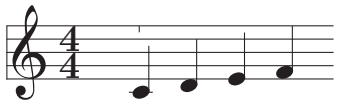
Duration. The duration of the musical note can be defined as the time interval between a note onset and the end of its decay. This attribute refers to the length of a musical note. Written Western music uses a system of notes, time signatures and tempo to define the length of a note. For an example, let us consider a musical notation of the first four notes of the C major scale where the tempo is defined as ♩ = 120. Western music notes are constructed to fit within the time signature defined. We will not discuss this in detail as it is a vast study outside the scope of this research.

TABLE I. MIDI VELOCITY AND DYNAMICS MAPPING IN LOGIC PRO X (RETRIEVED FROM [23]).

Dynamics notation	*ppp*	*pp*	*p*	*mp*	*mf*	*f*	*ff*	*fff*
MIDI Velocity	16	32	48	64	80	96	112	127



A 𝅗𝅥 represents a value of $1/2$, a ♩ represents a note value of $1/4$, and a ♪ represents a value of $1/8$ within the $4/4$ time signature. The tempo is defined above as ♩ = 120 BPM (beats per minute), meaning 120 repetitions of a ♩ can occur within a span of 60 seconds (1 minute). We can then deduce that a 𝅗𝅥 will have a duration of 1 second, a ♩ o will have a duration of 0.5 seconds, and a ♪ will have the duration of 0.25 seconds.

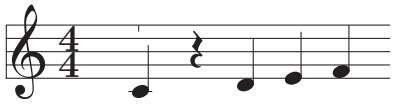
S= 

The four notes presented above are the first four notes of the C major scale starting with middle C. Earlier subsections elaborate on the MIDI note values, which in this case are, $C_4 = 60$, $D_4 = 62$, $E_4 = 64$, and $F_4 = 65$. Let us assume that the tempo is ♩= 120 BPM, and that the amplitude is a constant, medium loud value ($\boldsymbol{mf} \approx 80$). The earlier explanation shows us that the duration of a ♩ is 0.5s for a tempo of 120 BPM.

Using the representation we have introduced above (see equation 1), we can denote these four notes as follows:

$$S = \begin{bmatrix} 60 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 62 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 64 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 65 \\ 80 \\ 0.5 \end{bmatrix}. \quad (2)$$

The MIDI Standard defines a *note_on*, and *note_off* value for each musical note. The *note_on* is used to denote the starting value of each note and any pauses that may be present before the note. The *note_off* value denotes the end of each note, and we use a manipulation of these values to denote the note duration and pauses. A pause is denoted by a note with 0 amplitude and 0 pitch. Below is a musical notation of sequence $\tilde{S}$, which was obtained by introducing a 0.5s pause to S, immediately after its first note.

$\tilde{S}$= 

Equation 3 shows $\tilde{S}$, presented in the representation standard we have defined in equation 1. It should be noted that MIDI note number 0 is used to denote the C_{-1} note, as described earlier. The note $C_0 = 16.35Hz$ is typically cited as being just below the lower limit of human hearing [24], [25], [26]. As C_{-1}, is an octave below C_0, it is well outside the frequency range of most, if not all, commonly used musical instruments. Therefore, for our application, we can safely rule out the chances of a C_{-1} note occurring, and use its representation as a pause instead.

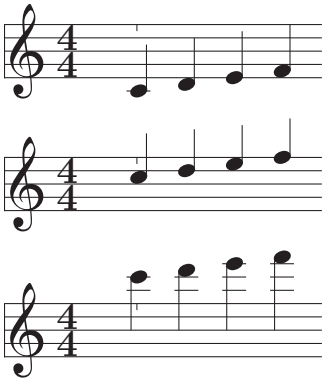
$$\tilde{S} = \begin{bmatrix} 60 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 62 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 64 \\ 80 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 65 \\ 80 \\ 0.5 \end{bmatrix}. \quad (3)$$

IV. Patterns and Tolerances

Our study is primarily geared towards live music applications and, therefore, we need to take the human tolerances into account (in terms of playing). It is unlikely that a musical performance will go on *exactly* at the notated tempo and, hence, the duration of notes may slightly vary. The dynamics might also vary depending on the performer, his interpretation or his interactions with the audience. For instance, the musician might play a solo section with different dynamics to better engage with his audience. In such a case, the system needs be aware of any allowable tolerances with respect to a pattern. We have defined tolerances for each of the note attributes as follows:

- Pitch - The tolerance for pitch is 0. This is due to the fact that any change in a MIDI note's pitch may completely transform the pattern. As Western music works under the principle of the 12-TET tuning system, any change in pitch by the lowest possible interval (semitone), will change the musical note.
- Amplitude - We set the tolerance for amplitude to approximately $6\% - 8\%$. This value corresponds to 8-10 units of deviation of the MIDI velocity, and is consistent with approximately 50% of the distance between music dynamics markers [23].
- Duration - It is not unlikely that the tempo may vary from what is stated due to human errors and various other uncontrollable factors, but too much of a variation will obstruct in experiencing the music as it was meant to be. It is rather unlikely that a piece of music meant to be played at a certain tempo (e.g., ♩ = 120 BPM), will be played at a drastically different tempo (e.g., ♩ = 220 BPM). With this understanding, we defined the tolerance for note duration as 5% of the tempo. This approach is sensible in the way that most music might become technically demanding at higher tempi, and the susceptibility to errors may be higher. In such a case, the tolerance becomes more lenient than at lower tempi.

Another characteristic of the 12-TET tuning system is the presence of octaves. For the purposes of this study, we consider that any identical sequences of notes, at different octaves, are different from each other. Refer to the musical notation below:



All three staves above show the identical sequence in all aspects except for their octave. In this study, we will consider these three sequences as three different patterns. This approach is made possible by the MIDI note number mapping.

V. Overview of Proposed Methods

In this study, we explore various methods to detect patterns. These methods can be categorized into two broad categories as Probabilistic and Deterministic approaches. As discussed in the previous section, one of the fundamental challenges of this study is the error tolerances and the octave-variant pattern representation. The task is indeed a classification task, but establishing the boundaries between classes proved difficult as the sequence of notes can be spread throughout the space, and various notes can be a part of a sequence with a defined order of occurrence.

Let us consider the sequence given by Eq. 2, which has four notes of various pitch. If we now consider the first note of sequence S, along with the tolerances that we have defined above, the note can lie in the following ranges:

$$S_0 = \begin{bmatrix} P_0 = 60 \\ 70 \leq A_0 \leq 80 \\ 0.475 \leq D_0 \leq 0.525 \end{bmatrix}.$$

A clearer understanding could be obtained if the note was viewed in a 3-dimensional space with each attribute being an axis (refer to Fig. 1). Here the x-axis represents the pitch, the y-axis represents the amplitude and the z-axis represents the duration. Fig. 1 shows a single note, with exact values, which can be plotted as a single dot. If we assign the tolerance values for the note, there will be a range of values that may deem the note as acceptable for recognition. Fig. 2 shows such a case. $\hat{S}_0$ denotes the note S_0 and the specified tolerances. As we have defined a zero tolerance for the pitch, we can visualize the range of values belonging to $\hat{S}_0$ as a rectangle centered around S.

Further to this, let us consider the sequence $\hat{S}$, which denotes S and its tolerances in a 3-dimensional space. Fig. 3 shows the set of values belonging to all 4 notes of S, within their tolerances. We can clearly see the difficulty in establishing inter class boundaries when trying to classify multiple sequences.

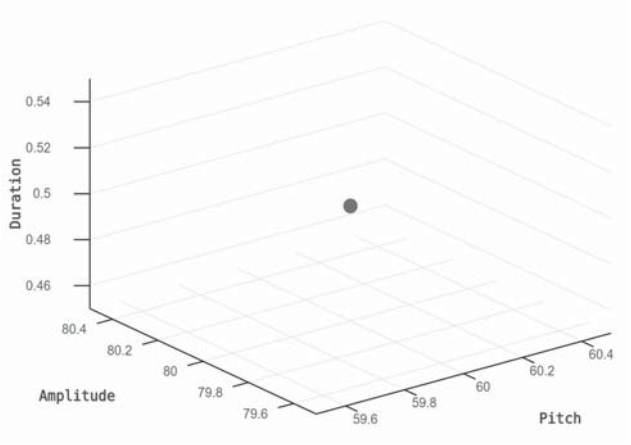


Fig. 1. A single note S_0 represented as a point in a 3-dimensional space

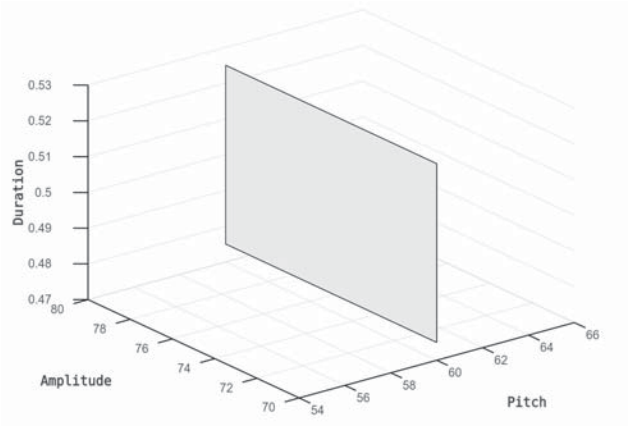


Fig. 2. A single note $\hat{S}_0$, with its tolerances represented in a 3-dimensional space

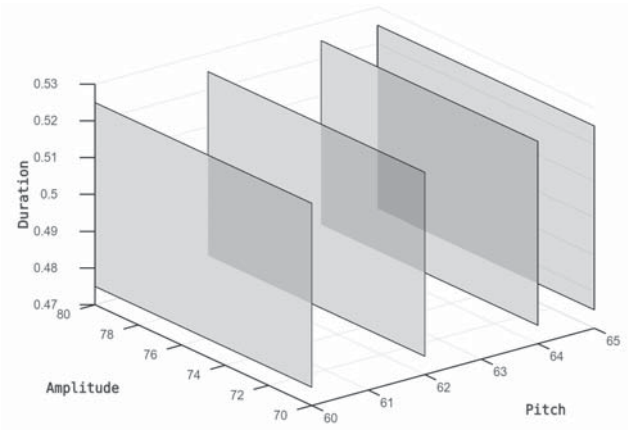


Fig. 3. The sequence $\hat{S}$, with its tolerances, represented in a 3-dimensional space

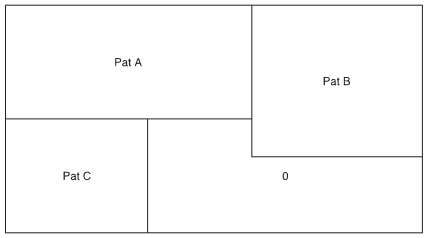


Fig. 4. Example of boundaries of three patterns

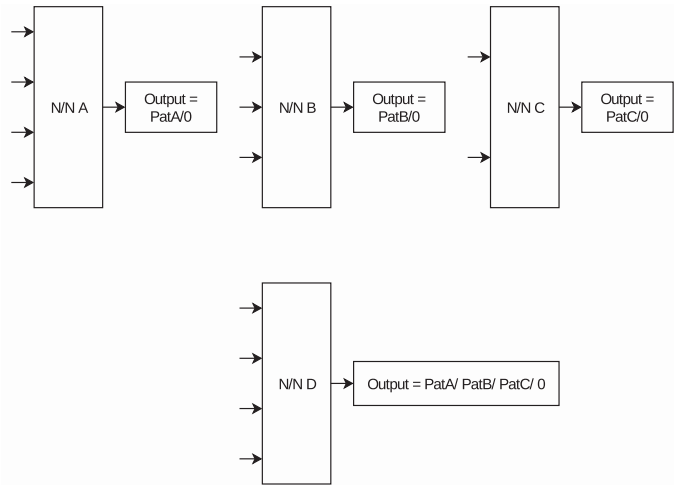


Fig. 5. An illustration of three individual neural networks, each designed to detect a single pattern (top), and a single neural network designed to detect all three patterns (bottom).

A. Deterministic Approach

We present a deterministic boundary checking system to detect patterns. This method represents a set of cascading conditional statements where each state acts as a memory. We can illustrate this method by Fig. 4, where the universal set denotes all the possible sequences of notes that can happen. If the sequence lies between the boundaries corresponding to Pat A, the system will return as such. If the sequence does not belong to any pattern set, the system will return a 0.

We are presenting a simple, boundary checking algorithm, which allows each note in the sequence to lie within specified tolerances. A detailed description of the algorithm is presented in the Section VI-A.

B. Probabilistic Approaches

We present six probabilistic methods of pattern identification. These methods will return the probability of a match against each pattern class. We have explored four methods based on artificial neural networks (ANN), a method based on convolutional neural networks (CNN), and a method based on recurrent neural networks (RNN). The ANN-based methods consist of a single neural network-based configuration and a multiple neural network-based configuration (see Fig. 5). Both configurations have two variants: a variant with a single hidden layer, and one with eight hidden layers.

The top section of Fig. 5 shows an overview of a multiple neural network based method. There are three neural networks,

each trained to detect a single pattern. In Fig. 5 top section, NN_A has been trained to detect a 4-note pattern. Similarly, NN_B and NN_C have been designed to detect patterns of lengths 3 and 2, respectively. The output of each neural network is binary, denoting a match to its corresponding pattern or not, and each neural network will take an input of a different length. The bottom section of the figure shows a single neural network-based approach. The input size is constant and shorter patterns will have to be padded and length equalized. The output of the single neural network denotes the probability of the input sequence match with each pattern.

VI. Methodology

This section presents the algorithms and detailed descriptions of each of the methods we have explored in this study.

A. Deterministic System

For ease of explanation, let us consider a sample pattern $PT = pt_1,\ pt_2,\ pt_3,\ pt_4$. Let the incoming sequence be $S_i = s_0,\ s_1,\ s_2,\ ...,\ s_m$, where m is the length of the stream (which is currently unknown). The amplitude and duration tolerances are defined as ΔA and ΔD. Note that the i^{th} note of the input sequence and of the pattern is expressed respectively as s_i and p_i for ease of representation.

In reality, each note s_i and pt_i will denote all attributes as illustrated in Eq. 1, where P_{s_i} and P_{pt_i} denote the pitch of s_i and pt_i, A_{s_i} and A_{pt_i} denote the amplitude of s_i and pt_i, and D_{s_i} and D_{pt_i} denote the duration of s_i and pt_i respectively, as shown below.

$$s_i = \begin{bmatrix} P_{s_i} \\ A_{s_i} \\ D_{s_i} \end{bmatrix},\ p_i = \begin{bmatrix} P_{pt_i} \\ A_{pt_i} \\ D_{pt_i} \end{bmatrix}$$

Algorithm 1 Deterministic System

1: For every incoming note S_i
2: Check if $s_i = pt_1$. For this condition to suffice, all attributes of s_i should be within the tolerances of p_i in such a way that
   - $P_{s_i} = P_{pt_1}$
   - $(A_{pt_1} - \Delta A) \leq A_{s_i} \leq (A_{pt_1} + \Delta A)$
   - $(D_{pt_1} - \Delta D) \leq D_{s_i} \leq (D_{pt_1} + \Delta D)$
3: If $s_i = pt_1$, check if $s_{i+1} = pt_2$, if not, increment i and go back to previous step;
4: If $s_{i+1} = pt_2$, check if $s_{i+1} = pt_2$;
5: If $s_{i+2} = pt_3$, check if $s_{i+3} = pt_4$;
6: The pattern PT is found if:
   - $s_i = pt_1$ and
   - $s_{i+1} = pt_2$ and
   - $s_{i+1} = pt_2$ and
   - $s_{i+3} = pt_4$

B. Single Neural Network

As explained earlier, this approach utilizes a single neural network trained to identify all within-tolerance variations of given patterns. One of the biggest constraints in such an approach is the fixed size of the input layer, and that patterns will not always have the identical length.

We overcome this limitation by padding the shorter patterns to obtain a list of patterns that have equal lengths. In our simulations, we constructed a padding note pd with values outside the sample space for all note attributes. The padding note is:

$$pd = \begin{bmatrix} 999 \\ 999 \\ 999 \end{bmatrix}$$

The shorter patterns are padded with pd to create a set of patterns with the same length. These are then used to create the training dataset.

Algorithm 2 Single neural network, training dataset creation

Inputs: *List of patterns, Labels*

1: Identify the longest pattern
2: Pad the shorter patterns with pd to equalize length.
3: Replace all padded notes with random notes that allow the best representation of the sample space. Label with the relevant pattern label.
4: Create within-tolerance variations for all patterns. For each pattern note pt_i, the newly created pattern variation note v_i should fall within the tolerances:
   - $P_{v_i} = P_{pt_1}$
   - $(A_{pt_i} - \Delta A) \leq A_{v_i} \leq (A_{pt_i} + \Delta A)$
   - $(D_{pt_i} - \Delta D) \leq D_{v_i} \leq (D_{pt_i} + \Delta D)$
5: Create amplitude variations of all patterns.
6: Create negative patterns that lie outside the tolerances of the given patterns.

Once the training dataset is created, we create and train the neural network. The patterns are flattened to obtain a single row vector to be input to the neural network. The input layer will have the same length as the flattened longest pattern, and the output layer will have a number of nodes equal to the number of patterns + 1, to account for negative patterns.

The system will save each incoming note in memory until a sub-sequence of required length is obtained. This sub-sequence is used to obtain a pattern prediction. Upon the arrival of each new note, the sub-sequence will discard its first note, and append the new note at its end. This operation is equivalent to the application of a sliding window on a discrete time series, where the window is shifted by 1 at each time increment. The real-time operation of the single neural network-based algorithm is highlighted in Algorithm 3.

Algorithm 3 Single neural network, operation

Input: *Music sequence* $S_i = s_0,\ s_1,\ S_2,\ ...,\ s_m$.

1: Store each incoming note in memory, until a sub sequence of the required length in obtained;
2: Reshape the sub sequence to obtain a single row;
3: Obtain the pattern prediction through the trained neural network;
4: Output the prediction.

C. Multiple Neural Networks in parallel

As explained in the above section, we train a single neural network per pattern. Such as approach is made possible due to the limited number of patterns. In this method, for n patterns, there will be n neural networks and n training datasets. Each neural network will have a binary output as it only needs to predict if the pattern is a match or not. Algorithms 4 and 5 explain the training dataset creation and the operation of this method.

Algorithm 4 Multiple neural networks, training dataset creation

Inputs: *List of patterns, Labels*

1: Create within-tolerance variants for each pattern, as elaborated in algorithm 2, and label them as positive;
2: Create amplitude variations for each positive patterns, label them as positive;
3: Create negative patterns that lie outside the tolerances of the given patterns, and label them as such;
4: Repeat for each pattern.

Algorithm 5 Multiple neural networks, operation

Input: *Music sequence* $S_i = s_0,\ s_1,\ s_2,\ ...,\ s_m$.

1: Construct sub sequences $ss_{i(i=1,2,..,n)}$ of lengths $[l_{pt_1}, l_{pt_2}, ..., l_{pt_n}]$, where l_{pt_i} denotes the length of pattern pt_i, and n denotes the number of patterns;
2: Pass each sub sequence ss_i to Neural Networks MNN_i, where l_{ss_i} is equal to the MNN_i's input layer length;
3: Reshape the sub sequence to obtain a single row;
4: Obtain the pattern prediction through the trained neural networks;
5: Output the prediction.

D. Convolutional and Recurrent Neural Networks

The training dataset creation for these methods are similar to those explained in algorithms 2 and 4. The operating procedure is very similar too. Sub-sequences extracted from the audio data stream is passed to each neural network.

Fig. 6 illustrates the operation of the two neural network based methods. The top section shows a multiple neural network based method with 3 networks each with an input size of 4, 3, and 2. The numbered line represents the incoming musical notes at each time increment. It can be clearly seen that the first 4 note sub-sequence of the audio stream is passed to NN_A, the first 3 note sub-sequence is passed to NN_B, and the first 2 note sub-sequence is passed to NN_C. Each network will output 1 if the input sub-sequence is a match to the pattern it was trained to detect, or 0 if otherwise.

The bottom section of figure 6 Shows the operation of a single neural network trained with an input layer size of 4 notes. Each consecutive 4 note long sub-sequence is extracted and input to the NN_D, which outputs the probability of matching with each pattern it was trained with.

VII. Dataset and Results

To observe the performance of the proposed methods, we used several fabricated data streams, and MIDI transcriptions

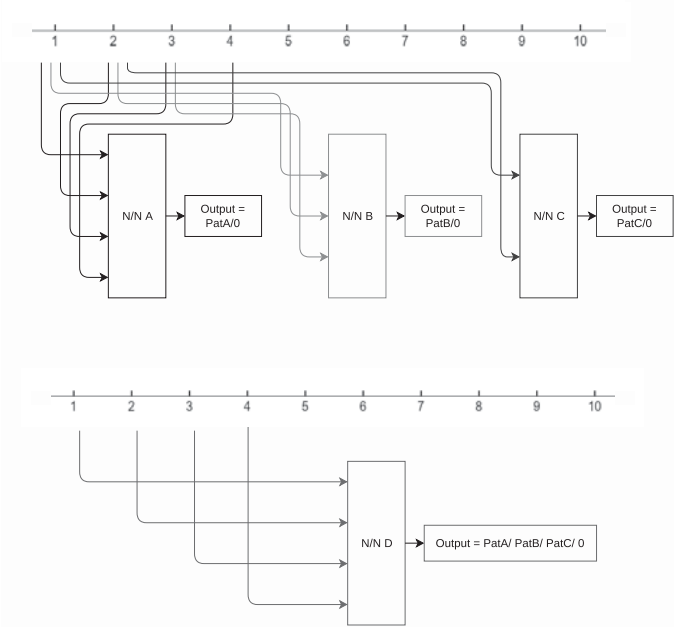


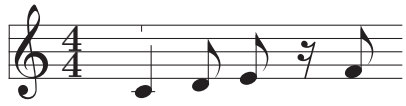
Fig. 6. The two neural network configurations, and an illustration of the sub-sequences extracted through the incoming data stream (represented by the numbered line).

of numerous popular songs and musical pieces. As this study deals with monophonic input sources, the data was chosen carefully to ensure that they are monophonic. The data and patterns are available online at [27]. The data and manually extracted patterns we used to train and evaluate the systems are as follows:

1) **Artificially created data stream 1**: 10 different patterns of various lengths were created manually. The patterns have lengths ranging from 4 to 10 notes. The data stream was created by placing each pattern consecutively. Each pattern is present once in the stream.
2) **Artificially created data stream 2**: 6 different patterns of varying lengths were created manually. Each pattern length is 5 notes. Some patterns have the same notes, but with a different ordering. A stream of 100 random notes were created, and the the generated patterns were placed at random locations. Some patterns are present multiple times in the stream.
3) **Flight of the bumblebee by Nikolai Rimsky-Korsakov**: 3 repeating sequences were chosen from the solo instrument track. Each sequence is 8 notes in length and repeats twice in the track.
4) **The solo guitar track from Scarified by Racer X**: Three patterns were manually extracted from the main melody. One pattern repeats 14 times throughout the song and has a length of 15 notes. Both other patterns are 16 notes in length, and repeat 3 times each. The solo track was obtained using the free resource [28].
5) **Canon in D by Johann Pachelbel**: The main melody was transcribed to MIDI. A 14 note segment was selected as a pattern. This segment occurs twice within the track.

6) **Frère Jacques**: A monophonic MIDI track of this French folk song was created and 2 short patterns were chosen. The patterns are 3 and 4 notes in length, and each pattern repeats twice.

To illustrate a sample pattern from a dataset, P_1 is one of the patterns used in *Artificially created data stream 1*. The musical notation as well as the column matrix based representation of P_1 is shown below:

$P_1 =$ 

$$P_1 = \begin{bmatrix} 60 \\ 127 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 62 \\ 60 \\ 0.25 \end{bmatrix}, \begin{bmatrix} 64 \\ 60 \\ 0.25 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0.25 \end{bmatrix}, \begin{bmatrix} 65 \\ 60 \\ 0.5 \end{bmatrix}.$$

The following methods were used for evaluation (all neural networks were designed using Python 3.7.3, TensorFlow 1.15.2, and Keras 2.3.1):

1) **Det:** Deterministic system;
2) **SNN-I:** Single network with a single hidden layer;
3) **MNN-I:**Multiple neural networks with a single hidden layer;
4) **SNN-II:** Single neural network with multiple hidden layers;
5) **MNN-II:** Multiple neural networks with multiple hidden layers;
6) **CNN:** Convolutional neural network;
7) **RNN:** Recurrent neural network.

The input layer length of ***SNN-I*** and ***SNN-II*** is equal to the pattern length. The output layer length is equal to the number of patterns + 1. Each neural network belonging to ***MNN-I*** and ***MNN-II*** has an input layer length corresponding to its pattern and an output layer length of 2. All input layers have a *ReLU* activation function and all output layers have a *softmax* activation function.

SNN-I and each neural network in ***MNN-I*** has one hidden layer, and uses a *ReLU* activation function. ***SNN-II*** and each neural network in ***MNN-II*** has 8 hidden layers and they too, use the *ReLU* activation function. All models used the *Adam* optimization algorithm.

CNN has one colvolutional layer with a *ReLU* activation function, immediately followed by a *max pooling* layer, and a *flatten* layer. ***CNN*** then employs a *dense* layer with a *ReLU* activation function, a *dropout* layer, and a *dense* output layer with a *softmax* activation function. The optimizer used by ***CNN*** is *Adadelta*.

RNN is equipped with two *LSTM* layer, a *dense* layer with a *ReLU* activation function, a *dropout* layer, and a *dense* output layer with a *softmax* activation function. The optimizer used by ***RNN*** is *Adam*.

Training datasets for each track were created using the chosen patterns as explained in algorithms 2 and 4. All methods were evaluated by mimicking a real-time music stream through the use of sliding windows, and each configuration was evaluated on the fly. All simulations were conducted on a workstation laptop with an Intel core i7 (2.8GHz) processor with 16GB of memory. Table II shows average running times for each method.

TABLE II. CONFIGURATIONS AND AVERAGE RUNNING TIMES FOR EACH INVESTIGATED METHOD.

Configuration	Running time (ms)
Det	2.1
SNN-I	18.5
MNN-I	35.2
SNN-II	15.8
MNN-II	38.6
CNN	19.5
RNN	24.8

TABLE III. CORRECT AND FALSE IDENTIFICATION PERCENTAGES FOR EACH INVESTIGATED METHOD.

Configuration	Correctly identified	Falsely identified
Det	100%	0%
SNN-I	90%	29%
MNN-I	92%	32%
SNN-II	96%	27%
MNN-II	97%	25%
CNN	98%	21%
RNN	99%	23%

The running times presented in table II relate to the time taken by each configuration to evaluate a single sub-sequence and was obtained by averaging the processing time per sub-sequence in each configuration. These values are presented in micro-seconds, and they prove that all methods are suitable for real-time applications.

Table III shows the results of the evaluations. The column *correctly identified* shows the number of sub-sequences where the system identified a pattern correctly, as a percentage of the total number of pattern containing sub-sequences. The *falsely identified* column shows the number of non-pattern containing sub-sequences the system registered as a pattern, as a percentage of the total number of sub sequences where no pattern is present.

VIII. DISCUSSION

The results of the experiments show that, while all configurations have high rates of detection, the probabilistic models tend to have some false positives. The boundary checking system seems to have the best performance due to its deterministic nature. However, if we take into account a real-world scenario, there is a chance that a musician might play a required pattern slightly outside the specified tolerance values. And in such a case, the deterministic system will recognize it as a non-pattern. One could say that due to the deterministic nature of this method, it may become too artificial and strict. We cannot expect a musician performing their instrument to play an exact copy of the musical piece each and every time. The subtle nuances and idiosyncrasies is the very thing that makes music so enjoyable. Along that thought, one could argue that machine learning methods may form hidden neural pathways that are potentially able to capture these nuances. Therefore, probabilistic methods may work better than strict rule-bound deterministic systems and overcome the case mentioned above. However, future research is needed to further reduce the rate

of false positives, and subsequently prepare such a system for live real-time use.

Various opportunities exist to extend the results of our research. A possible extension of this paper concerns the combination of two or more of the discussed methods. As explained in earlier sections, it is extremely difficult to construct class boundaries due to the nature of the pattern note positions. This is another prospective extension of this work. Notably, the present study was conducted considering exclusively monophonic music signals. The possible application of the methods presented here on polyphonic single track music, monophonic multi track music, and polyphonic multi track music would also render this study much more useful to real world applications. Furthermore, this work applied the proposed methods on a mobile workstation computer, whereas smart musical instruments rely on embedded systems. In future work we plan to investigate how to manage the computational complexity and make it possible to put the methods developed for musical pattern recognition into actual smart instruments, as well as to devise novel IoMusT applications based on them.

IX. Conclusions

In this paper we investigated musical patterns detection in real-time, a topic that has not yet received adequate attention in the field of Music Information Retrieval, and which is relevant to several IoMusT applications. Specifically, we considered the case of a monophonic streams of MIDI notes as produced in real-time by a smart musical instrument. For this purpose, we presented a representation mechanism to denote musical notes as a single column matrix, whose contents correspond to three key attributes of each musical note - pitch, amplitude and duration. Based on such representation, we compared the most prominent candidate methods based on neural networks and one deterministic method. Numerical results show the accuracy of each method, and allow us to characterize the trade-offs among those methods.

We believe that this study is among the first in the area of real-time musical pattern detection for IoMusT applications, and several developments are needed to address the numerous research directions and potentialities.

References

[1] E. Margulis, "Musical repetition detection across multiple exposures," *Music Perception: An Interdisciplinary Journal*, vol. 29, pp. 377–385, 2012.

[2] Y. Lo and W. Li, "Linear time for discovering non-trivial repeating patterns in music databases," in *2004 IEEE International Conference on Multimedia and Expo*, 2004, pp. 293–296.

[3] J. Burgoyne, I. Fujinaga, and J. Stephen Downie, *Music Information Retrieval*. Wiley-Blackwell, Nov 2015, pp. 213–228.

[4] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and Challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2872625

[5] L. Turchet, "Smart Musical Instruments: vision, design principles, and future directions," *IEEE Access*, vol. 7, pp. 8944–8963, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2876891

[6] L. Turchet, A. McPherson, and M. Barthet, "Real-time hit classification in a Smart Cajón," *Frontiers in ICT*, vol. 5, no. 16, 2018. [Online]. Available: https://doi.org/10.3389/fict.2018.00016

[7] E. Cambouropoulos, "Musical parallelism and melodic segmentation," *Music Perception*, vol. 23, pp. 249–268, 2006.

[8] X. Gao, J. Zhang, and Z. Wei, "Deep learning for sequence pattern recognition," in *IEEE 15th International Conference on Networking, Sensing and Control*, 03 2018, pp. 1–6.

[9] A. Pikrakis, S. Theodoridis, and D. Kamarotos, "Classification of musical patterns using variable duration hidden markov models," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1795–1807, 2006.

[10] K. Yang and C. Shahabi, "A pca-based similarity measure for multivariate time series," in *Proceedings of the 2nd ACM International Workshop on Multimedia Databases*, 01 2004, pp. 65–74.

[11] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, p. 11, 01 1995.

[12] Z. Rafii and B. Pardo, "Repeating pattern extraction technique (repet): A simple method for music/voice separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, pp. 73–84, 2013.

[13] J.-L. Hsu and A. Chen, "Efficient repeating pattern finding in music databases." in *ACM CIKM International Conference on Information and Knowledge Management*, 01 1998, pp. 281–288.

[14] Y.-l. Lo and C.-y. Chen, "Fault tolerant non-trivial repeating pattern discovering for music data," in *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06)*, vol. 2006, 08 2006, pp. 130 – 135.

[15] K. Lemström, G. Wiggins, D. Meredith, T. Barn, M. Drove, and T. Giles, "Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music," *Journal of New Music Research*, 08 2002.

[16] Y. Zhu, H. L. Tan, and S. Rahardja, "Drum loop pattern extraction from polyphonic music audio," in *2009 IEEE International Conference on Multimedia and Expo*, 08 2009, pp. 482 – 485.

[17] Shazam. [Online]. Available: https://www.shazam.com. [Accessed: 05.06.2020].

[18] A. Wang, "An industrial strength audio search algorithm." in *4th International Conference on Music Information Retrieval*, 01 2003.

[19] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, and G. Toussaint, "Computing a geometric measure of the similarity between two melodies," in *15th Canadian Conference on Computational Geometry*, 09 2003.

[20] A. Lubiw and L. Tanur, "Pattern matching in polyphonic music as a weighted geometric translation problem." in *5th International Conference on Music Information Retrieval*, 01 2004.

[21] B. Haynes, *A History of Performing Pitch: The Story of 'A'*. Scarecrow Press, 2002.

[22] D. M. Randel, *The Harvard Dictionary of Music*. Harvard University Press Reference Library, 2003, vol. 4.

[23] Logic Pro X: Use step input recording techniques. [Online]. Available: https://support.apple.com/kb/PH12977. [Accessed: 05.06.2020].

[24] International Pitch Notation. [Online]. Available: http://www.flutopedia.com/octave_notation.htm. [Accessed: 05.06.2020].

[25] H. F. Olson, *Music, physics and engineering*. Dover Publications New York, 1967, vol. 2.

[26] N. Silva, P. C. Weeraddana, and C. Fischione, "On musical onset detection via the s-transform," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1080–1085.

[27] Dataset and patterns. [Online]. Available: https://github.com/ns2max/real_time_patt. [Accessed: 09.06.2020].

[28] Ultimate Guitar. [Online]. Available: http://ultimate-guitar.com. [Accessed: 05.06.2020].