Project Assignment – Bio-inspired Computing

Pouya Mohseni – 610398164

Genetic Algorithm for Jazz Improvisation: Introducing a Fitness Function for Evaluating Jazz Improvisation

Abstract

In this assignment, we explore part of the literature on generating jazz solos for a given tune using a genetic algorithm. Our goal is to introduce and enhance the reviewed techniques used for this purpose.

To begin, we review and implement the existing genetic algorithms in the jazz solo generation. We explore the genome structure and discuss the disadvantages faced by the methods.

Moreover, we propose an enhancement to the current genetic algorithms employed. By introducing a Conditional Variational Autoencoder (CVAE), we incorporate anomaly detection techniques into the evaluation process. Subsequently, we survey various music representations and their suitability for use in neural-based networks. Afterward, we present a graph-based representation of jazz improvisation.

In the table of contents below, we have highlighted our contributions, including improvements and discussions.

Link to the repository: https://github.com/PouyaMohseni/JazzImprov

Table of Content

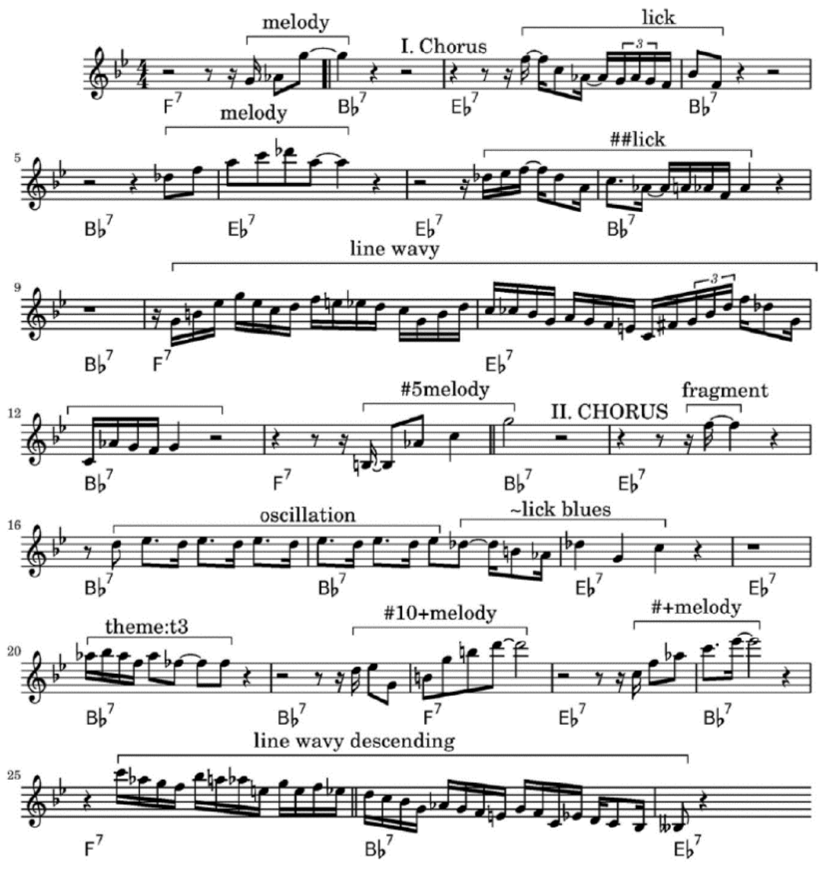
Introduction

Jazz improvisation is the spontaneous creation of music, incorporating the artist's creativity and allowing for unique expressions. These features allow musicians to produce pieces of music that stand apart from the others.

However, improvisation is not always without constraints. In many jazz genres, players consider some kind of unwritten roles. Some of the musicians adhere to the chord progression. Some might disregard harmony in many cases. Moreover, the tune and well-known licks play an important part in the improvisation.

Generating jazz music using genetic algorithms is beneficial due to the inherent randomness and crossover of successful patterns. This approach ensures that each generated piece possesses its own distinct characteristics, promoting diversity and preventing repetitive outcomes. It enables innovation and exploration, resulting in rich and creative jazz compositions.

However, the improvisation problem is hard to be simulated with a genetic algorithm having so many possibilities of notes and chords that do not usually appear in true compositions. Therefore, we simplify the problem by only considering monophonic jazz improvisation. Therefore, the generated piece can be adapted by saxophone, trumpet, or any other monophonic instrument. Additionally, we aim to generate bebop improvisation which emerged in the 1940s.



Genetic Algorithm for Jazz Soloes (GenJam)

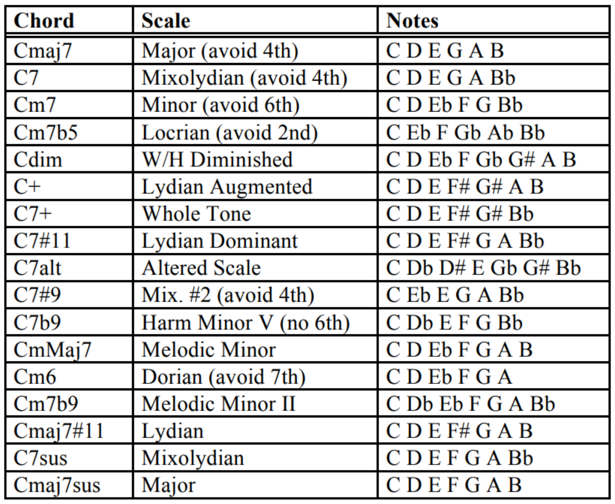
In this section, we delve into the highly acclaimed GenJam genetic algorithm developed by John Biles, as introduced and discussed in his papers [1][2][3]. Notably, the first paper has garnered over 800 citations in the literature.

2.1. Chromosome Representation

Like all genetic algorithms, the choice of representation scheme is critical to the efficiency if not the efficacy of the resulting system. Briefly, a tune consists of a sequence of choruses; a chorus is a sequence of phrases; a phrase is a sequence of measures; and a measure is a sequence of events. The chromosomes of the problem consist of two-level phrases and sequences. The evolution operates on the phrase and measure levels, evolving populations for each level in parallel.

The Measure Population consists of 64 individuals, each of which decodes to a measure of eighth-note length events. Rest events (coded as 0) are performed as a MIDI note-off event. Hold events (coded as 15) are performed by doing nothing, which results in the prior event being held through the hold event's time window. New-note events (coded as 1-14) are performed as a MIDI note-off followed by a MIDI note-on, using the event value as an offset into roughly two octaves of the scale suggested by the chord being played by the rhythm section at the time the event is performed.

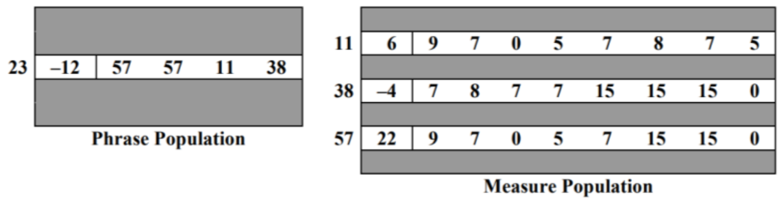
To ensure that only safe notes are used in a chromosome 17 families of chords GenJam recognizes, along with the scales it uses for each chord type, using C as an example root tone is introduced as follows:

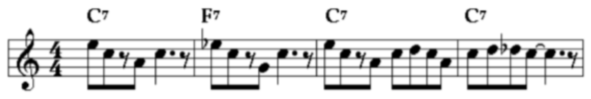


Chord	Scale	Notes
Cmaj7	Major (avoid 4th)	C D E G A B
C7	Mixolydian (avoid 4th)	C D E G A Bb
Cm7	Minor (avoid 6th)	C D Eb F G Bb
Cm7b5	Locrian (avoid 2nd)	C Eb F Gb Ab Bb
Cdim	W/H Diminished	C D Eb F Gb G# A B
C+	Lydian Augmented	C D E F# G# A B
C7+	Whole Tone	C D E F# G# Bb
C7#11	Lydian Dominant	C D E F# G A Bb
C7alt	Altered Scale	C Db D# E Gb G# Bb
C7#9	Mix. #2 (avoid 4th)	C Eb E G A Bb
C7b9	Harm Minor V (no 6th)	C Db E F G Bb
CmMaj7	Melodic Minor	C D Eb F G A B
Cm6	Dorian (avoid 7th)	C D Eb F G A
Cm7b9	Melodic Minor II	C Db Eb F G A Bb
Cmaj7#11	Lydian	C D E F# G A B
C7sus	Mixolydian	C D E F G A Bb
Cmaj7sus	Major	C D E F G A B

The population consists of 48 individuals, each of which decodes into four pointers to measures in the measure population. In other words, GenJam utilizes four-measure phrases.

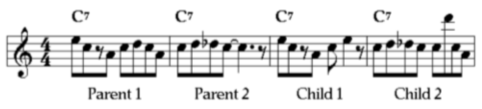
Below, an example of a four-measure phrase is illustrated. The number in the upper left corner represents the fitness value, followed by the phenotype of the represented chromosome:





2.2. crossover

The crossover operator behaves identically in both the measure and phrase populations. One point crossover is performed to parents and results in two children. This operation is done in binary representation and as a result, might lead to the destruction of an event. Below, a crossover between two measures is illustrated:



The Measure Population consists of 64 individuals, each of which decodes to a measure of eighth-note length events. Rest events (coded as 0) are performed as a MIDI note-off event. Hold events (coded as 15) are performed by doing nothing, which results in the prior event being held through the hold event's time window. New-note events (coded as 1-14) are performed as a MIDI note-off followed by a MIDI note-on, using the event value as an offset into roughly two octaves of the scale suggested by the chord being played by the rhythm section at the time the event is performed.

2.3. Mutations

After crossover, one of the two children is selected at random to be mutated using one of GenJam's musically meaningful mutation operators. In this case, the "mentor" cannot listen to hundreds or thousands of generations of a soloist – a dozen or two generations is about the limit. This puts enormous pressure on the mutation operators to come up with new individuals that are not just different but clearly better than their parents.

2.3.1. Measure Mutations

The mutation operators for measures implement adaptations of several standard melodic development techniques – transposition, retrograde, rotation, inversion, sorting, and retrograde-inversion. Transpose adds a random fixed number to each new note event. Reverse, reverses the events in the measure. Rotate right, moves the events to the right with a fixed random number.

Invert, and subtracts each event from 15 (rest becomes holds). Sorts, hold the rest, and hold events and sort notes in an ascending or descending order. Invert/reverse, performs both invert and reverse operators. The table below illustrates the output of the operators for a simple genome:

Original Measure	9	7	0	5	7	15	15	0
Transpose (up 3)	12	10	0	8	10	15	15	0
Reverse	0	15	15	7	5	0	7	9
Rotate right (3)	15	15	0	9	7	0	5	7
Invert	6	8	15	10	8	0	0	15
Sort up	5	7	0	7	9	15	15	0
Sort down	9	7	0	7	5	15	15	0
Invert/Reverse	15	0	0	8	10	15	8	6

2.3.1. Phrase Mutations

Reverse and rotate operators are defined as the same as the measure mutation operators. The sequence Phrase operator picks a random measure pointer in the phrase chromosome and replaces it with one of its neighbors. The Genetic Repair operator replaces the least fit measure in the phrase with a randomly selected one. The Lick Thinner operator attacks the convergence problem by replacing the measure in the phrase that occurs most frequently in the entire phrase population with a measure that occurs infrequently. The Orphan Phrase operator takes this idea to an extreme by creating a brand-new phrase of the four measures that occur least frequently in the entire phrase population.

Original Phrase	57 57 11 38
Reverse	38 11 57 57
Rotate Right (3)	57 11 38 57
Sequence Phrase	57 57 ***38*** 38
Genetic Repair	57 57 11 ***29***
Super Phrase	41 16 57 62
Lick Thinner	***31*** 57 11 38
Orphan Phrase	17 59 43 22

The aforementioned operators are implemented in the file named “GenJam - Genetic Algorithm for Jazz Soloes.ipynb”. Moreover, the examples in the paper have been simulated in the file. Notice that because of the stochastic behavior the output does not always adhere to the paper examples.

2.4. Selection

Breeding new generations in both populations begins with a tournament selection/replacement procedure, which randomly chooses four individuals from the population to form a “family.” The two fittest members of the family are selected to survive and become parents, and their two children replace the two weakest family members back in the population. This proceeds until 50% of each population is replaced. The fitness values for the children are initialized to zero, and they are not allowed to participate in subsequent families in the same generation because they have not been “heard” yet by the mentor. After evaluating a new generation, the mentor decides whether the soloist is good enough to play in public. If not, another generation can be bred.

2.5. Fitness

The Mentor is essentially the human evaluation function that indirectly provides fitness values for the individuals in both populations. This makes GenJam a classic interactive genetic algorithm (IGA), which is often necessary in artistic applications, due to the difficulty in developing algorithms that can determine artistic merit.

2.6. Discussion

Based on the representation of the chromosome and the construction of the crossover and mutations, we can identify some disadvantages of this algorithm.

2.6.1. Rhythmic limitation

GenJam is restricted rhythmically to eighth-note multiples. This limitation is a result of the inherent constraints in the representations of the chromosomes. More complex structures would provide the capability to generate and simulate a wider variety of rhythmical patterns.

2.6.2. Outside playing limitation

One drawback of GenJam's inability to play "wrong" notes is its limitation in breaking harmonic rules in the creative and unexpected ways that skilled human soloists often do. This reflects a fundamental design choice, prioritizing competent playing over brilliant yet potentially discordant performances. Breaking these rules can indeed sound remarkable in the right context, but it typically requires extensive experience and taste to avoid sounding incompetent.

2.6.3. Measure limitation

Furthermore, the constraint of four measures may not suffice for many musical sections. In numerous jazz compositions, sections are structured in 8 or 16 measures; as a result, GenJam may not be capable of generating a complete jazz solo in these cases.

2.6.4. Lick limitation

Some jazz musicians incorporate well-known licks into their playing, facilitating a connection with jazz listeners. However, in this algorithm, this approach is not utilized.

2.6.5. Sequence-playing limitation

While the algorithm does address sequence playing, it approaches the task in a somewhat oversimplified manner. Sequence playing encompasses various forms that cannot be adequately addressed within this structure.

2.6.6. Melody adherence limitation

The adherence to the original melody is a feature that can be included in the design of jazz improvisation algorithms. Some models aim to capture the essence of jazz by respecting the melody and harmonies from the original tune. However, it's crucial to assess the capabilities and design choices of a specific algorithm to determine whether or not it accounts for melody adherence.

2.6.7. Scale adaptability limitation

In Jazz music, the adaptability of musical phrases across different scales is a crucial aspect of improvisation. Skilled jazz musicians often explore the versatility of their musical ideas by moving them between scales. The algorithm being examined in this context efficiently handles phrase transportation within scales of the same tonality.

However, an inherent limitation becomes evident when dealing with scales that have varying numbers of "consonant" notes. It's a common scenario in jazz that not all notes within a scale are created equal, and their suitability can depend on the specific harmonic context. As a result, transporting a melody from a scale with, for instance, six "consonant" notes to one with seven may pose challenges, and vice versa. This issue is particularly prevalent when navigating between scales with different intervals, which can affect the compatibility of musical phrases.

2.6.8. Scale-chord relation

In the realm of jazz music, the relationship between chords and scales is multifaceted and does not adhere to a strict one-to-one correspondence. This paper primarily focuses on the most widely used scales within the context of each chord. However, it's essential to acknowledge that the versatility of jazz allows for the exploration of numerous scales that can harmoniously complement a given chord, contributing to a viable and creative solo. The flexibility and richness of jazz improvisation are evident in its ability to embrace a wide array of scale choices, enhancing the expressive possibilities for musicians.

2.6.9. Evaluation

Given that the mentor is required to assess all measures and phrases in a real-time harmonic context, a substantial bottleneck in terms of fitness evaluation exists. In contrast to Image-Generating Algorithms (IGAs) that can process individuals in parallel or in a condensed format, GenJam's individuals cannot be efficiently evaluated in a similar manner. The mentor must meticulously listen to each individual one at a time, in real-time, while accompanied by suitable harmonic context. This task can be particularly challenging for most individuals. Furthermore, when dealing with large populations, the mentor faces the demanding task of listening to a significant number of individuals in each generation to provide fitness values for all.

A Genetic Algorithm for the Generation of Jazz Melodies

In this section, we explore the chromosome and fitness function introduced in the paper [4]. Although this paper offers a genetic algorithm for generating jazz melodic, the chromosome structure and the fitness function can be adapted in the jazz improvisation generation.

3.1. Chromosome Representation

Like the GenJam algorithm, the representation in this paper follows a scale-based method. Therefore, out-scale playing is still restricted. However, it adapts the rhythmic characteristics of each note, therefore, it enables more rhythms that are dividable by four. The chromosome is then a sequence of <extended-degree, duration pairs>, rests being distinguished by the constant rest in place of the extended-degree.

3.2. Fitness

Unlike the GenJam approach, in this paper, a knowledge-based fitness function is introduced. The fitness function evaluates eight distinct characteristics of a chromosome, from which it calculates, via a weighted sum, the corresponding overall fitness. Although this criterion is used for jazz melodies, some parts of it can be adapted to jazz improvisation.

3.2.1. Large intervals

The user can specify the largest permissible interval between consecutive notes. The fitness penalty is the sum of the sizes of the intervals that are larger than permissible, multiplied by a constant weight.

3.2.2. Pattern matching

In this implementation, pattern matching occurs only between pitch fragments, and not rhythmic ones. The output of the pattern-matching algorithm is a list of numbers, where each number denotes that there exist two similar patterns of length in the melody. The system does not attempt to find overlapped similar patterns. By default, the patterns must be five or more notes in length.

3.2.3. Suspensions

The paper considered four cases, with distinct weights: there is a consonant suspension, meaning that the note is a member of both scales determined by the two consecutive chords; there is a dissonant suspension, which means that the note is a member of the first scale but not of the second; there is no suspension; or there is a rest.

3.2.4. Note at downbeat

The first beat in a bar is usually the most musically significant beat in that bar which this characteristic emphasizes on that.

3.2.5. Note at half-bar

It is the same as the downbeat function but for the third beat of each bar. However, the emphasis is lower due to the weakness of the beat.

3.2.6. Long notes

The user can specify what she considers to be long notes. Long notes are mostly used in music as points of stasis. Therefore, it is preferable to have harmonically stable long notes.

3.2.7. Contour

This is a comparison between the contour of the chromosome and the contour specified by the user. The user specifies whether the average pitch of each bar is lower than, the same as, or higher than the last.

3.2.8. Speed

Similar to Contour, except that the algorithm makes an estimate of the speed of the piece simply by adding the number of notes and rests in each pair of consecutive bars, and matching them to 'slow', 'medium', or 'fast' as appropriate.

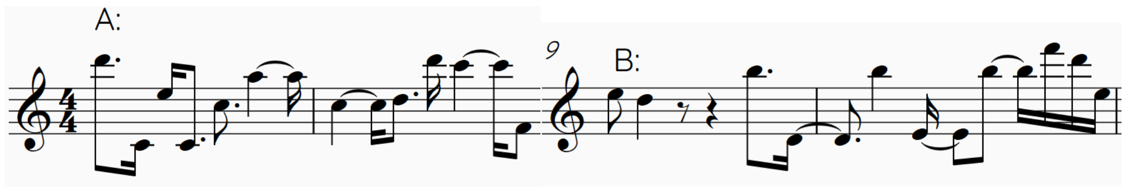
The chromosome representation of this algorithm along with the adapted fitness function of this paper can be found in "A Genetic Algorithm for the Generation of Jazz Melodies.ipynb".

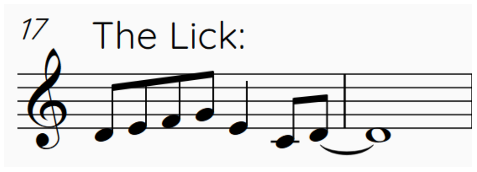
3.3. Discussion

We have covered the implementation of this algorithm in the mentioned .ipynb file. In addition, we made efforts to adapt the fitness function to provide a more meaningful evaluation for jazz improvisation. The adapted fitness function takes into account factors such as note intervals, the number of events in downbeats and half bars, the presence of long notes, and the identification of matched patterns for consideration.

Furthermore, to test the effectiveness of the introduced fitness function, we conducted a comparison between two randomly generated improvisations – with initialization aligned with the paper's parameters – and one of the most renowned and recognized jazz licks. The following are the two initialized tunes for this comparison:

```
Improvisation Tune:  A
Degrees:   [15, 1, 10, 1, 8, 19, 8, 9, 16, 15, 4]
Durations: [3, 1, 1, 3, 3, 5, 5, 3, 1, 5, 2]
-----------------------------------------------------------------------------------------
Improvisation Tune:  B
Degrees:   [10, 9, 0, 0, 14, 2, 14, 20, 14, 18, 16, 10]
Durations: [2, 4, 3, 3, 3, 4, 4, 3, 3, 1, 1, 1]
-----------------------------------------------------------------------------------------
```





We continued by calculating the fitness of these improvisations utilizing the introduced function:

```
Fitness of A:  0
Fitness of B:  -2
Fitness of The Lick:  0
```

Clearly, the fitness function proves to be inadequate in distinguishing between a randomly generated improvisation and the most famous lick. This limitation arises due to the involvement of numerous evaluation criteria, with the ultimate preference leaning towards the improvisation that best fits the intended musical context and is, above all, most appreciated by the listener.

Introducing a Fitness Function Based on Anomaly Detection for Jazz Improvisation

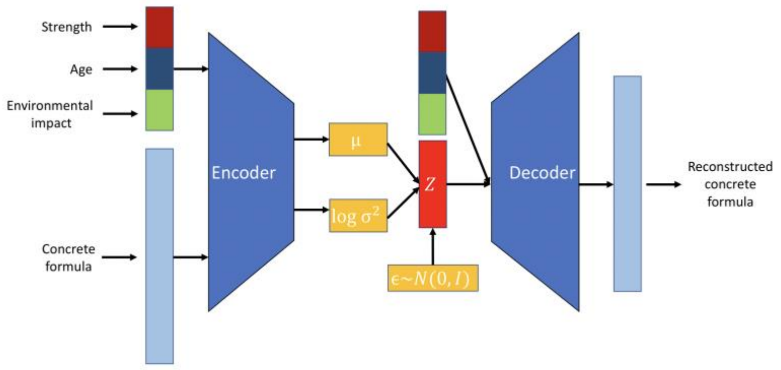
4.1. Introduction and Definition

In this section, we introduce a new approach to the fitness function through deep learning. The presented model is based on CVAEs and it is trained with the jazz improvisations from various artists on different tunes. Consequently, the model is able to give a score to any of its input regarding its closeness to the training improvisations.

As far as we are concerned, no research has utilized Anomaly detection in jazz music or improvisation generation or in any other music generation algorithms. Therefore, we consider this a new approach to defining the fitness function. However, there are works in music genre classification utilizing anomaly detection.

Anomaly detection is a technique used to identify unusual patterns or outliers in a given dataset. Autoencoders, a type of neural network, are commonly employed for anomaly detection. They consist of an encoder and a decoder, working together to learn a compressed representation of the input data. When trained on normal samples, autoencoders can reconstruct them accurately. However, when presented with anomalies, the reconstruction error tends to be higher, indicating their anomalous nature.

On the other hand, Variational Autoencoders (VAEs) are generative models that combine autoencoders with probabilistic techniques. VAEs can generate new samples by sampling from a learned latent space. By encoding the data into a probability distribution, VAEs offer more flexibility in generating diverse and realistic outputs. Conditional Variational Autoencoders (CVAEs) extend VAEs by incorporating additional conditional information during training and generation, allowing for targeted generation based on specific conditions [6].



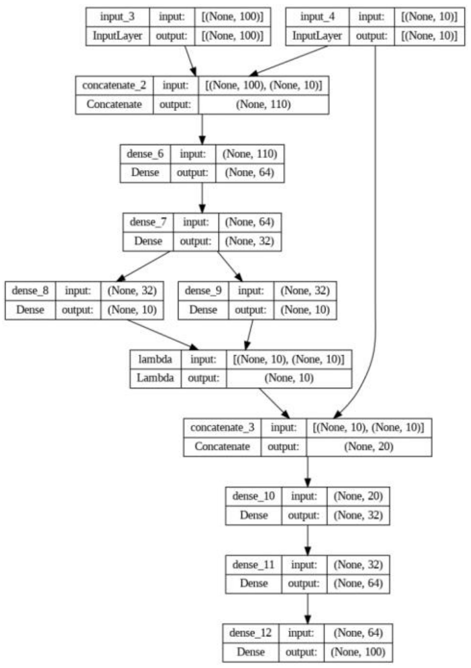
Our motivation for introducing this model is rooted in three key ideas:

1) Music, particularly jazz improvisation, exhibits inherent randomness; when we request a musician to perform an improvisation, it will inherently differ from the original rendition, while conveying the same values and meaning.
2) Certain musical features should be learned, such as the capacity to introduce rests, while others can aid in comprehending a piece, like understanding chord progressions or the artist's theme.
3) Improvisation includes features, such as pitch variance or the presence of rest notes, which derive from a probability distribution. We aim to harness this distribution's learnable aspects via the variational layer.

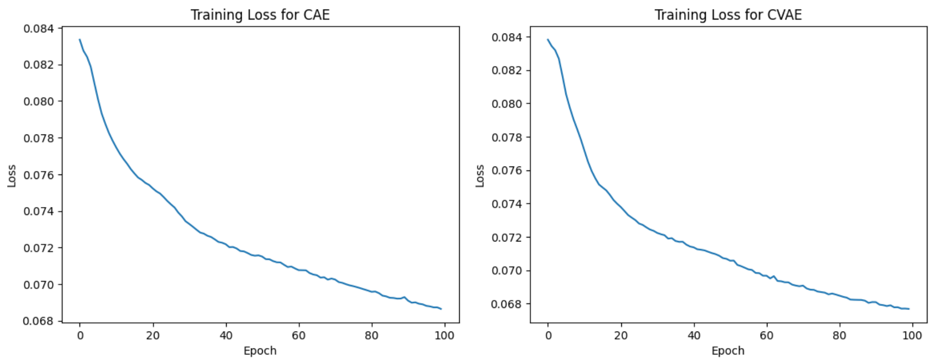
Thus, we try to train the CVAE using improvisations by renowned artists across various tunes. The acquired model subsequently gains the capability to distinguish between authentic jazz improvisation and random noise. Assuming that the characteristics of improvisation follow a distribution, we can employ this model as our fitness function. During the testing phase, we simply input the chromosome and calculate the accuracy. The higher the accuracy, the better the model fits the given improvisation.

4.2. Conditional Variational Autoencoder Implementation

We used the Keras library in order to implement CVAE. In the file "Conditional Variational Autoencoder Implementation.ipynb" we first constructed a conditional autoencoder with dense layers with a condition size of 10 and an input size of 100. After that, we added a sampling function between the encoder and the decoder in order to present the CVAE. This is the illustration of the introduced model with 3 decoder and 3 encoder dense layers:


input_3 | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
input_4 | input: | [(None, 10)]
InputLayer | output: | [(None, 10)]
concatenate_2 | input: | [(None, 100), (None, 10)]
Concatenate | output: | (None, 110)
dense_6 | input: | (None, 110)
Dense | output: | (None, 64)
dense_7 | input: | (None, 64)
Dense | output: | (None, 32)
dense_8 | input: | (None, 32)
Dense | output: | (None, 10)
dense_9 | input: | (None, 32)
Dense | output: | (None, 10)
lambda | input: | [(None, 10), (None, 10)]
Lambda | output: | (None, 10)
concatenate_3 | input: | [(None, 10), (None, 10)]
Concatenate | output: | (None, 20)
dense_10 | input: | (None, 20)
Dense | output: | (None, 32)
dense_11 | input: | (None, 32)
Dense | output: | (None, 64)
dense_12 | input: | (None, 64)
Dense | output: | (None, 100)

We, then, trained both models with randomly generated data in order to assess their learning abilities. The loss curve of the training phases for both models is illustrated below:



4.3. Chromosome Representation

The last piece of the puzzle is, perhaps, the representation of the improvisation. A good representation decreases the learning time and might enhance the encoder's latent layer's representation efficiency. The problem to consider is that the model should be able to work with the representation for every (or at least most of the) piece(s). In the following, we survey some of the representations and introduce graph representation for music pieces.

4.3.1. GenJam Representation

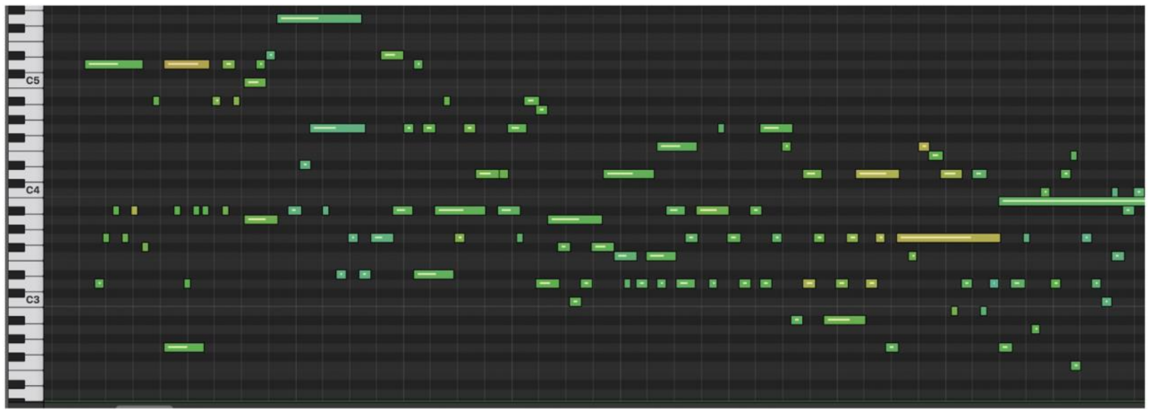
The GenJam representation is discussed in the second section. Although this representation has a lot of restrictions, if an improvisation can be represented by this criterion, it can be fed to any introduced model. This is because the length of the chromosomes is always the same.

4.3.2. Timestep Representation

The under-discussed representation is introduced in the third chapter. Unlike the GenJam representation, the length of the chromosome in this representation is proportional to the number of notes in the improvisation. Thus, if the model has dense layers, this representation is ill-suited because the first layer of the model has a fixed length.

4.3.3. Midi Representation

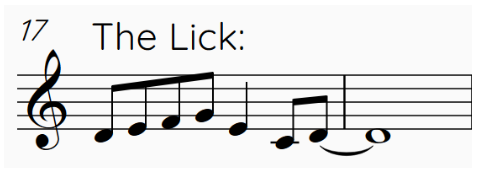
Midi representation is another widely-used representation of music. In many studies, the music is turned into a midi file. Thus, the representation of the music is analogous to a photo. Therefore, CNN-based models can be used in the process of learning. For example, in our case, we might change the dense layers to convolution or pooling layers. Here is a midi representation of a tune:

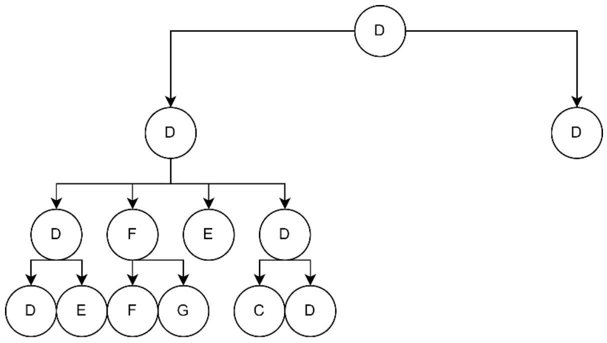


The advantages of this model are the easy implementation and the fixed length of the input. However, the training phase will be costly due to the inefficiency of the representation.

4.3.4. Graph Representation

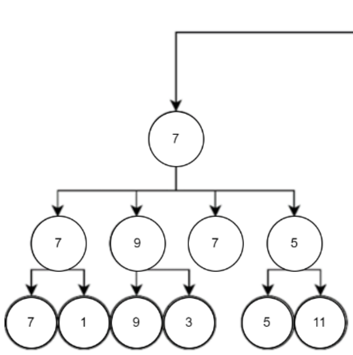
Due to the growth of the use of graphs in studies and the advances in graph-based deep learning, we introduce a music representation based on graphs. Each piece is represented by a tree based on duration-importance. To put it in a nutshell, the tree begins with a root node, representing the piece itself. The root has 'n' children, each of which signifies one of the 'n' measures that compose the music. Within each measure, there are 'k' children, representing the 'k' beats contained in that measure. Furthermore, each beat can be subdivided into twos, threes, or fives to account for different note durations. This division continues until a leaf node represents a specific note. Subsequently, we assign names to each leaf node, and these note names are propagated upwards to their parent nodes. Each parent's name is allocated based on the name of the leftmost child.





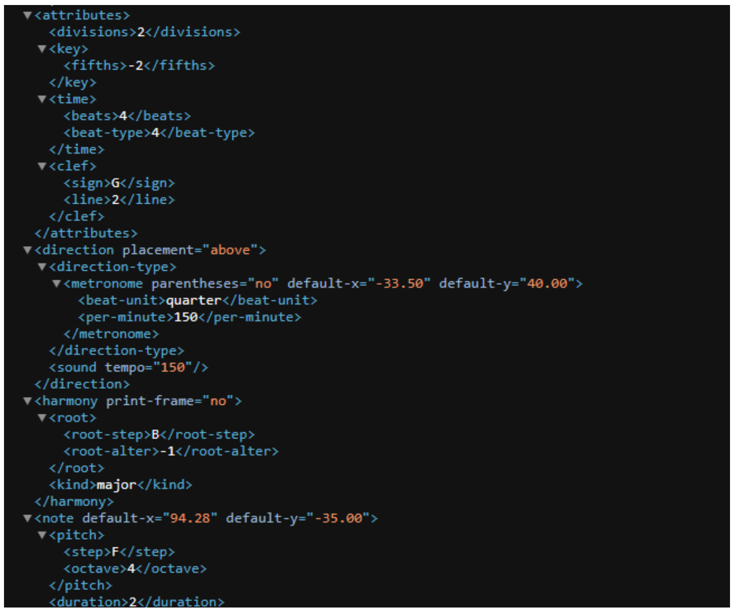
As can be seen, this representation offers a lightweight graph for every tune. Thus, a graph-based model can learn features of jazz improvisation easily. However, the representation can fail to represent some tunes correctly. But, with a small manipulation, every tune can be represented.

To enhance the representation's effectiveness, we can incorporate the mode of each section into our representation. This can be achieved by considering relative pitch, rather than absolute pitch, in relation to the underlying mode. For example, when dealing with 'the lick' and considering chord changes from Em7 to Am7 to Dm7, we can create a representation that allows us to interpret a single note like 'D' in different ways based on the underlying mode. In this case, 'D' is the 7th, 5th, and 1st note from the root in each mode, respectively.



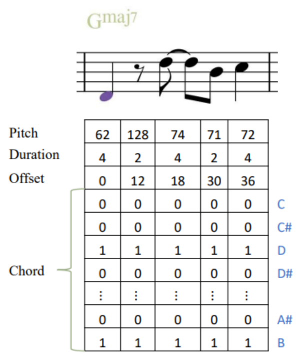
By employing the graph representation as chromosomes, we simplify the process of evaluating and evolving jazz solos. Each chromosome encapsulates the structure and characteristics of a jazz solo in a manner that can be effectively processed by the algorithm. The graph-based nature of the representation enables the algorithm to capture the nuanced relationships between different notes, durations, and modes.

Trying to derive this representation from an image input is hard, therefore, we rely on MusicXML. MusicXML is a file format used to store and exchange music notation data digitally. It allows different music software and platforms to communicate and share sheet music, enabling composers, musicians, and music enthusiasts to collaborate, edit, and display musical scores accurately. A part of the MusicXML format of the piece “I Got Rhythm” is shown below:



```
▼<attributes>
   <divisions>2</divisions>
  ▼<key>
     <fifths>-2</fifths>
   </key>
  ▼<time>
     <beats>4</beats>
     <beat-type>4</beat-type>
   </time>
  ▼<clef>
     <sign>G</sign>
     <line>2</line>
   </clef>
 </attributes>
▼<direction placement="above">
  ▼<direction-type>
    ▼<metronome parentheses="no" default-x="-33.50" default-y="40.00">
       <beat-unit>quarter</beat-unit>
       <per-minute>150</per-minute>
     </metronome>
   </direction-type>
   <sound tempo="150"/>
 </direction>
▼<harmony print-frame="no">
  ▼<root>
     <root-step>B</root-step>
     <root-alter>-1</root-alter>
   </root>
   <kind>major</kind>
 </harmony>
▼<note default-x="94.28" default-y="-35.00">
  ▼<pitch>
     <step>F</step>
     <octave>4</octave>
   </pitch>
   <duration>2</duration>
```

In the paper which the Bebopnet [7] is introduced, another representation is captured. This representation is based on events and is adapted for monophonic jazz improvisation:



Pitch	62	128	74	71	72	
Duration	4	2	4	2	4	
Offset	0	12	18	30	36	
Chord	0	0	0	0	0	C
	0	0	0	0	0	C#
	1	1	1	1	1	D
	0	0	0	0	0	D#
	⋮	⋮	⋮	⋮	⋮	
	0	0	0	0	0	A#
	1	1	1	1	1	B

In which:

Duration: The duration of each note is encoded using a one-hot vector consisting of all the existing durations in the dataset. Durations smaller than 1/24 are removed.
Offset: The offset of the note lies within the measure and is quantized to 48 “ticks” per (four-beat) measure. This corresponds to a duration of 1/12 of a beat. This is similar to the learned positional encoding used in translation.
Chord: The chord is represented by a four-hot vector of size 12, representing the 12 possible pitch classes to appear in a chord. As common in jazz music, unless otherwise noted, we assume that chords are played using their 7th form. Thus, the chord pitches are usually the 1st, 3rd, 5th, and 7th degrees of the root of the chord. This chord representation allows the flexibility of representing rare chords such as sixth, diminished, and augmented chords.

However, this representation can be used as a pivot in order to transform MusicXML into the graph representation. The code which extracts events representation from the XML can be found at “gather_data_from_xml.py”.

4.4. Feature Architecture

In this section, we determine the feature architecture of the fitness model. Moreover, we discuss why certain features are fed to the encoder while others are used as conditions.

4.4.1. Encoder’s Input

The input to the encoder is an important component for determining the fitness of improvisation; the fitness decoder should be able to reconstruct the improvisation based on the encoded features in the middle layer. In this model, this input is the graph representation of the improvisation itself.

4.4.1. Condition

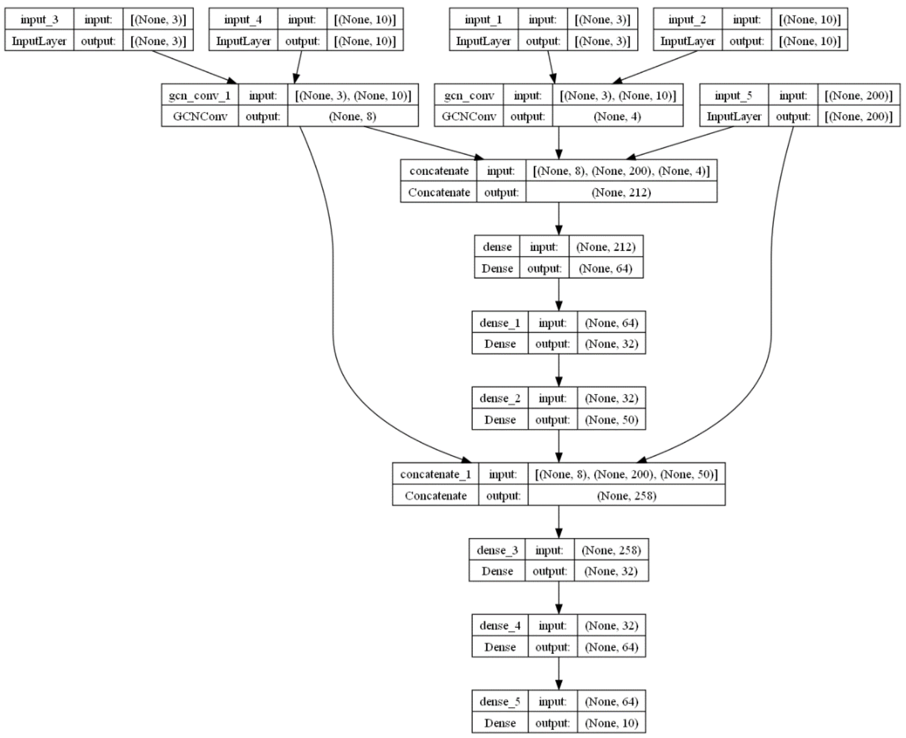
Other relevant data which can help in order to evaluate an improvisation form the condition of the model. One of the important factors for evaluating the improvisation is the chord progression of the tune. We consider the improvisations to be 16 fix bars with each bar having at most two chords. Therefore, the chord progression of the model can be encoded in an array with fix length of 17*32 with the decoding table below:

Chord	Scale	Notes
Cmaj7	Major (avoid 4th)	C D E G A B
C7	Mixolydian (avoid 4th)	C D E G A Bb
Cm7	Minor (avoid 6th)	C D Eb F G Bb
Cm7b5	Locrian (avoid 2nd)	C Eb F Gb Ab Bb
Cdim	W/H Diminished	C D Eb F Gb G# A B
C+	Lydian Augmented	C D E F# G# A B
C7+	Whole Tone	C D E F# G# Bb
C7#11	Lydian Dominant	C D E F# G A Bb
C7alt	Altered Scale	C Db D# E Gb G# Bb
C7#9	Mix. #2 (avoid 4th)	C Eb E G A Bb
C7b9	Harm Minor V (no 6th)	C Db E F G Bb
CmMaj7	Melodic Minor	C D Eb F G A B
Cm6	Dorian (avoid 7th)	C D Eb F G A
Cm7b9	Melodic Minor II	C Db Eb F G A Bb
Cmaj7#11	Lydian	C D E F# G A B
C7sus	Mixolydian	C D E F G A Bb
Cmaj7sus	Major	C D E F G A B

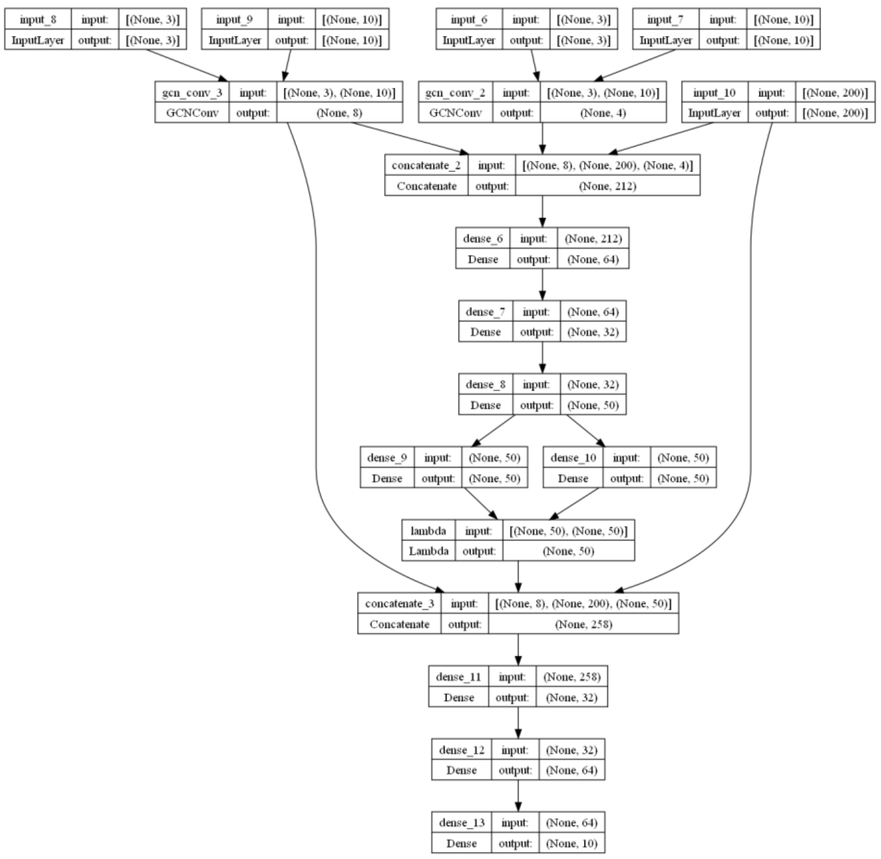
Another contributing element to the evaluation is the adherence of the improvisation to the melody. Referring to some parts of the melody during the improvisation most of the time is apricated. Therefore, we try to make the melody available for the evaluation process. The melody, also, can be represented using the introduced graph representation. Thus, a graph representation is also passed to the model.

4.5. Graph Conditional Variational Autoencoder Implementation

We used the Keras library in order to implement GCVAE. In the file “Graph Conditional Variational Autoencoder Implementation.ipynb”. We used the structure of the previously introduced model and changed it in a way that it could accept a graph as its input. There are three different inputs for this model: 1) the graph representation of the improvisation, 2) the graph representation of the melody, and 3) the vector representation of the chord progression. While the first two inputs are in the graph representation, we first imply a graph convolution layer in order to change them to meaningful vector representations. This is the introduced GCAE illustration:


input_3 | InputLayer | input: [(None, 3)] | output: [(None, 3)]
input_4 | InputLayer | input: [(None, 10)] | output: [(None, 10)]
input_1 | InputLayer | input: [(None, 3)] | output: [(None, 3)]
input_2 | InputLayer | input: [(None, 10)] | output: [(None, 10)]
gcn_conv_1 | GCNConv | input: [(None, 3), (None, 10)] | output: (None, 8)
gcn_conv | GCNConv | input: [(None, 3), (None, 10)] | output: (None, 4)
input_5 | InputLayer | input: [(None, 200)] | output: [(None, 200)]
concatenate | Concatenate | input: [(None, 8), (None, 200), (None, 4)] | output: (None, 212)
dense | Dense | input: (None, 212) | output: (None, 64)
dense_1 | Dense | input: (None, 64) | output: (None, 32)
dense_2 | Dense | input: (None, 32) | output: (None, 50)
concatenate_1 | Concatenate | input: [(None, 8), (None, 200), (None, 50)] | output: (None, 258)
dense_3 | Dense | input: (None, 258) | output: (None, 32)
dense_4 | Dense | input: (None, 32) | output: (None, 64)
dense_5 | Dense | input: (None, 64) | output: (None, 10)

With the use of the introduced variational layer, we construct a GCVAE shown as below:


input_8 | InputLayer | input: [(None, 3)] | output: [(None, 3)]
input_9 | InputLayer | input: [(None, 10)] | output: [(None, 10)]
input_6 | InputLayer | input: [(None, 3)] | output: [(None, 3)]
input_7 | InputLayer | input: [(None, 10)] | output: [(None, 10)]
gcn_conv_3 | GCNConv | input: [(None, 3), (None, 10)] | output: (None, 8)
gcn_conv_2 | GCNConv | input: [(None, 3), (None, 10)] | output: (None, 4)
input_10 | InputLayer | input: [(None, 200)] | output: [(None, 200)]
concatenate_2 | Concatenate | input: [(None, 8), (None, 200), (None, 4)] | output: (None, 212)
dense_6 | Dense | input: (None, 212) | output: (None, 64)
dense_7 | Dense | input: (None, 64) | output: (None, 32)
dense_8 | Dense | input: (None, 32) | output: (None, 50)
dense_9 | Dense | input: (None, 50) | output: (None, 50)
dense_10 | Dense | input: (None, 50) | output: (None, 50)
lambda | Lambda | input: [(None, 50), (None, 50)] | output: (None, 50)
concatenate_3 | Concatenate | input: [(None, 8), (None, 200), (None, 50)] | output: (None, 258)
dense_11 | Dense | input: (None, 258) | output: (None, 32)
dense_12 | Dense | input: (None, 32) | output: (None, 64)
dense_13 | Dense | input: (None, 64) | output: (None, 10)

Discussion

The introduction of a fitness model based on anomaly detection using a CVAE and a graph representation of jazz improvisations allows a better evaluation and generation of jazz solos. This approach utilizes the inherent randomness in jazz music and the crossover and combination features of music composition, aiming to capture the characteristics that define jazz improvisation.

The choice of representation is crucial in this approach, as it directly impacts the model's ability to learn and evaluate jazz improvisations. The graph representation offers a lightweight and flexible way to represent jazz solos. This allows neural layers to understand the underlaying relationship of each layer in our hierarchy representation faster, allowing to introduce an online improvisation generator; it is important to consider the efficiency and effectiveness of the chosen representation in the context of the specific algorithm being used.

The use of a CVAE allows for conditional generation and the utilization of additional information, such as chord progressions and adherence to the melody. This enhances the model's ability to evaluate the improvisation from a more complete viewpoint, taking into account both the structural and melodic aspects of the music.

The evaluation of this fitness model can provide insights into its ability to distinguish between authentic jazz improvisation and random notes. The accuracy score is a key metric in assessing its performance. Nevertheless, the success of the model also depends on the diversity and quality of the training improvisations.

Overall, the introduction of a fitness model based on anomaly detection using a CVAE and a graph representation is a promising approach to jazz improvisation generation and evaluation. It addresses the inherent randomness and compositions of jazz music and offers a new perspective on how to assess the creativity of jazz solos.

Future Work

There are, in fact, several areas for improvement in the evaluation model. The proposed model relies on the simple message-passing layers available in the Graph Neural Network framework. However, these layers may not effectively capture and encompass hierarchical features in our representation. For instance, it requires the use of five layers of GCN message-passing to adequately capture the features of the last two notes in 'the lick' within the graph representation. Therefore, it is crucial to develop new layers that can capture features in the improvisation within a hierarchical structure. With the introduction of these layers, it should be possible to create a new representation at the graph's root in the encoded layer with just a few of them.

Moreover, the loss function needs to be improved and adopted to the hierarchical representation in order to have a valid assessment by the autoencoder. The loss function should be able to focus more on the important parts of the improvisation, dismissing small changes. We argue that the tree representation of the piece can, in turn, help to introduce this function as it inherently emphasizes the structural elements of the music.

References

[1] Biles, John. "GenJam: A genetic algorithm for generating jazz solos." ICMC. Vol. 94. 1994.
[2] Biles, John A. "GenJam: Evolution of a jazz improviser." Creative evolutionary systems. Morgan Kaufmann, 2002. 165-187.
[3] Biles, John. "Improvizing with Genetic Algorithms: GenJam." Evolutionary Computer Music (2007).
[4] Papadopoulos, George, and Geraint Wiggins. "A genetic algorithm for the generation of jazz melodies." Proceedings of STEP 98 (1998).
[5] Jordanous, Anna. "A fitness function for creativity in jazz improvisation and beyond." (2010).
[6] Ge, Xiou, et al. "Accelerated Design and Deployment of Low-Carbon Concrete for Data Centers." *ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies (COMPASS)*. 2022.
[7] Hakimi, Shunit Haviv, Nadav Bhonker, and Ran El-Yaniv. "BebopNet: Deep Neural Models for Personalized Jazz Improvisations." *ISMIR*. 2020.