UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Music Generation And Popularity Prediction Of Music Tracks Using Machine Learning Algorithms

Diploma Thesis

Maxouri Aikaterini

Supervisor: Tousidou Eleni

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Music Generation And Popularity Prediction Of Music Tracks Using Machine Learning Algorithms

Diploma Thesis

Maxouri Aikaterini

Supervisor: Tousidou Eleni

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δημιουργία Μουσικής Και Πρόβλεψη Δημοτικότητας Μουσικών Κομματιών Με Χρήση Αλγορίθμων Μηχανικής Μάθησης

Διπλωματική Εργασία

Μαξούρη Αικατερίνη

Επιβλέπουσα: Τουσίδου Ελένη

Βόλος 2021

Approved by the Examination Committee:

Supervisor **Tousidou Eleni**
Laboratory teaching staff, Department of Electrical and Computer Engineering, University of Thessaly

Member **Michael Vassilakopoulos**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Tsompanopoulou Panagiota**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 30-6-2021

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Dr. Eleni Tousidou for her valuable assistance in developing this thesis. I would, also, like to thank Dr. Michael Vassilakopoulos and Dr. Panagiota Tsompanopoulou for participating in the examination committee.

Last but not least, I would like to thank my family and my friends for always being there for me throughout my studies. Their support played a major role in accomplishing this thesis.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Maxouri Aikaterini
30-6-2021

Abstract

This thesis introduces an approach to automated music generation. More specifically, a neural network architecture is described which composes new music content. Its architecture is mostly comprised of Bidirectional Long Short-Term Memory layers, used to model the underlying structure of music. The model is trained in various datasets, containing pieces from different genres, trying to learn each genre's patterns. This approach managed to model music structure and generate harmonic pieces.

Περίληψη

Αυτή η διπλωματική εργασία παρουσιάζει μια προσέγγιση για αυτοματοποιημένη σύνθεση μουσικής. Πιο συγκεκριμένα, περιγράφεται η αρχιτεκτονική ενός νευρωνικού δικτύου που παράγει νέα μουσικά κομμάτια. Η αρχιτεκτονική του αποτελείται κυρίως από αμφίδρομα (bidirectional) Long Short-Term Memory επίπεδα, προκειμένου να μοντελοποιήσει τη βασική δομή της μουσικής. Το μοντέλο εκπαιδεύεται σε διάφορα σύνολα δεδομένων, τα οποία περιέχουν κομμάτια από διαφορετικά μουσικά είδη, με σκοπό να μάθει τα μοτίβα του κάθε είδους. Η προσέγγιση αυτή κατάφερε να μοντελοποιήσει επιτυχώς τη μουσική δομή στα περισσότερα είδη, όπως και να συνθέσει αρμονικά κομμάτια.

Table of contents

List of figures

List of tables

Abbreviations

AUC	Area under the ROC Curve
BPM	Beats Per Minute
CNN	Convolutional Neural Network
DL	Deep Learning
FPR	False Positive Rate
LSTM	Long Short-Term Memory
MIDI	Musical Instrument Digital Interface
NN	Neural Network
RBM	Restricted Boltzmann Machines
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
TPR	True Positive Rate

Chapter 1

Introduction

People have been interested in composing music since ancient times. Lately, automated music generation has been a popular topic of research [2][3][4][5][6][7][8]. In general, great accomplishments have been made by machine learning and deep learning study fields, such as text generation, speech recognition, speech generation etc. However, generating music with minimum human intervention is a challenge, due to the complex nature of music content. Any music piece is, basically, comprised of sequences of notes played by instruments. Nevertheless, those sequences are complicatedly structured and many aspects have to be considered by the composer in order to create a pleasing result. In automated music generation, two distinct probability distributions must be considered. The first one is the probability distribution of simultaneous notes, the distribution of chords in other words. The second one is the probability distribution of a note occurring right after a given note or sequence of notes. Last but not least, dynamics, that is the loudness of notes playing, must be considered, in order for the piece to be expressive rather than mechanical.

1.1 Related Work

There have been several approaches to automated music composition so far, all employing machine and deep learning techniques. Deep learning is preferred over models created manually, for instance grammar-based [9] or rule-based [10] music generation systems. The reason is that deep learning models implement an agnostic approach, meaning that they model the structure of data, without making the assumption that it can be modelled accurately. As a result, a deep learning model can be used for various music genres. Moreover, according to

[11] machine learning tools are able to create content, even when its nature is too complex to be modelled by formulations or brute force design, adding that they are more likely to generalize and perform well, in case inputs change.

To begin with, Todd [12] created the first artificial neural network for algorithmic composition in 1989. Todd, considering only the pitch and duration of each note, created a fully-connected network with also feedback connections, with the intention of learning musical structure. In addition, Mozer [13] extended the idea of learning notes sequentially and of modelling the probability of the next note given the previous sequence, by developing a recurrent auto-predictive connectionist network called CONCERT. This architecture uses representations not only of pitch and duration of notes, but also of harmonic structure. CONCERT, based on the content it generated, performed well at learning the underlying structure of music pieces. Those two approaches of algorithmic composition are fundamental for automated music generation, since many relative researches that conducted later, are relied on them.

Another approach is implemented by MiniBach model, a two layered feedforward neural network described in [2]. MiniBach is a music generation system which produces accompaniment pieces for alto, tenor and bass voices, provided a melody for a soprano voice. This system uses binary symbolic representation of music as input and output. Furthermore, the use of Restricted Boltzmann Machine (RBM) [14] combined with recurrent neural networks (RNN) was proposed by Boulanger-Levandowski et al. [15]. RBM models the distribution of chords by studying a musical corpus, while the recurrent network models the sequences of successive notes. As a result, the network can focus on two different dimensions of data, while there is interaction between them. To put it in simple words, the RBM focuses on the vertical dimension of music, that is simultaneous notes or chords. On the other hand, the RNN models the temporal sequence of notes in the horizontal dimension. The RNN-RBM network is able to produce harmonic polyphonic pieces, by combining the two distinct probability distributions. Due to their architectures though, both can produce only sequences of fixed length.

Oore et al. [16] created a Long Short-Term Memory (LSTM) neural network to generate music. They used symbolic representations of recorded human performances, in order to model dynamics of notes, adding that, by employing an iterative strategy, pieces' duration is not fixed. Their system generated classical music that lacked long-term structure, however the local structure, for example phrasing dynamics, was noticeable. Johnson [17], inspired by

the RNN-RBM architecture, proposed biaxial LSTMs to model probability distributions of both notes and chords. The network consists of two layers in the vertical axis and two in the horizontal, resulting in predictions on both axes. The results obtained showed that the model managed to generate consistent and harmonic pieces

Bidirectional recurrent networks are widely used in speech and text recognition together with generation [18] [19][20]. They have been shown to model accurately long-term dependencies, especially in combination with LSTM, since they are trained in both directions of sequences (from beginning to end and vice versa). They have also been implemented for composing music purposes [21] [22] [23] where the generated pieces sounded more pleasing to listeners, compared to pieces composed by recurrent networks or other approaches.

Yang et al. [24] proposed a convolutional generative adversarial network for generating pop accompaniment melodies. They employed Convolutional Neural Networks (CNN) as a generator for melodies, along with a discriminator to learn the distributions of the melodies. The final model, called MidiNet can generate multi-track pieces which sound realistic and pleasant, according to user study. For the same purpose, an encoder-decoder [25] architecture is suggested in PopMAG [26]. Based on the results, PopMAG modelled effectively harmony between the different tracks and long-term context. Finally, Jukebox [27] and WaveNet [28] are two of the very few models that consider music in the raw audio domain, represented by waveforms. They employed convolution feedforward networks to generate music of various genres along with vocal tracks.

1.2 Purpose of the Thesis

The purpose of this thesis is to create a music generating system, in order to generate multi-track music pieces. To elaborate more, the aim is to generate new and expressive melody pieces, as well as accompaniment pieces in respect to the melody ones. Most researches so far, are focused either on generating single-track music or on generating only accompaniment content in respect to already existing melodies. Furthermore, most of them consider binary representation of notes, meaning that loudness of notes are not engaged. However, this thesis aims to generate both original melodies and accompaniment tracks, while also considering note's loudness. To obtain harmonic and sound pleasing results, the system models the probability distributions of notes occurring simultaneously (chords), along with

the probability distributions of notes occurring after a given sequence of notes. Dynamics are also modelled, towards achieving expressiveness.

In addition, the system is trained in different genres to test its ability of learning different styles of music and producing content accordingly. Finally, since taste in music is subjective, quality of outcomes can not be evaluated in an objective way. This thesis aims to provide a more objective way of evaluation, by predicting the likelihood of a generated piece generated becoming popular in Spotify, a digital music service with a vast database of songs used by millions of users.

1.3 Organization of the Thesis

The thesis is organized as follows.

- Chapter 2 is a brief introduction to Deep Learning and its fundamental concepts employed for this thesis.
- Chapter 3 describes the format of data as well as the encoding implemented.
- Chapter 4 aims to analyze the models built for accomplishing music generation.
- Chapter 5 analyzes the results obtained.
- Chapter 6 is the conclusion of the thesis.

Chapter 2

Introduction to Deep Learning

The purpose of this chapter is to introduce Deep Learning (DL) and its basic concepts that were considered in this thesis.

2.1 Definition

Deep Learning is a subfield of machine learning, concerned with Neural Networks (NN). Neural Networks are series of algorithms inspired by the structure and function of human brain [29]. Basically, a NN architecture mimics the way a human brain operates in processing data, learning patterns and making decisions. A NN consists of a large set of units connected to each other. These units are also known as *neurons* or *nodes*.

A node in NN is a mathematical function that processes data according to network's architecture. Nodes are interconnected, adding that a weight is assigned to every connection link, which carries information about the input signal. Every node has also an internal state, also known as activation signal, produced by the combination of input and an activation function. A collection of nodes that operate together at a specific depth, is called *layer*.

2.2 Building blocks

The basic building blocks of neural networks are discussed in the following sections.

2.2.1 Topology

Topology of a network is the way its nodes as well as connection links are arranged. Based on topology, networks can be divided into two categories, the feedforward networks and the feedback ones.

Feedforward networks

Feedforward networks consist of consecutive layers. Every layer's nodes are connected to the nodes of the previous/next layer. In addition, there is no feedback loop hence information flows in one direction, from the input layer to the output. The first layer of the network is called *input layer*, the last layer is called *output layer*, while any layer in between is named *hidden layer*. In figure 2.1 the structure of a feedforward NN is depicted.

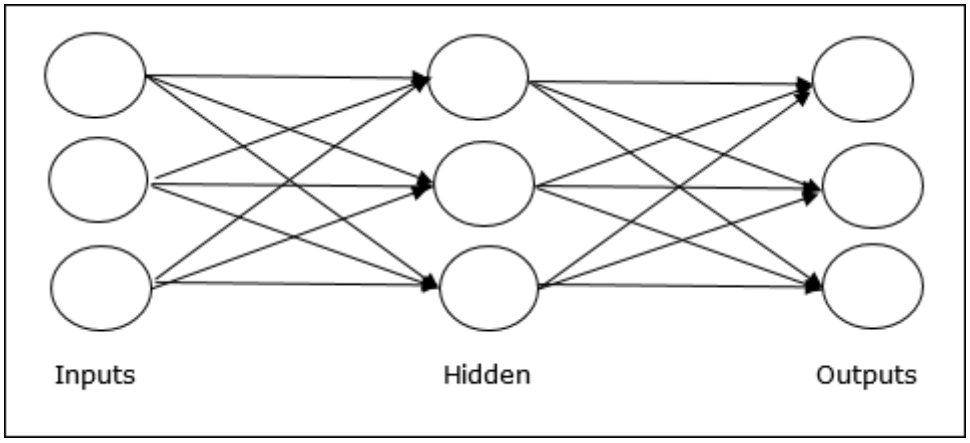

Figure 2.1: The structure of a feedforward NN. Reprinted from [1].

Feedback Networks

Feedback networks have also feedback loops, as a result information flows in both directions, from input to output as well as from output to input. In this thesis, recurrent neural networks (RNN) are considered, that is feedback networks with closed loops. An example of a fully recurrent architecture can be seen in figure 2.2, where all nodes are connected to all other nodes. Principally, a recurrent network is a feedforward one, extended with recurrent connections, in order to learn series of items. As a consequence, the output of a hidden layer reenters the layer as an input, hence the RNN learns not only from the current input, but also from its previous state.

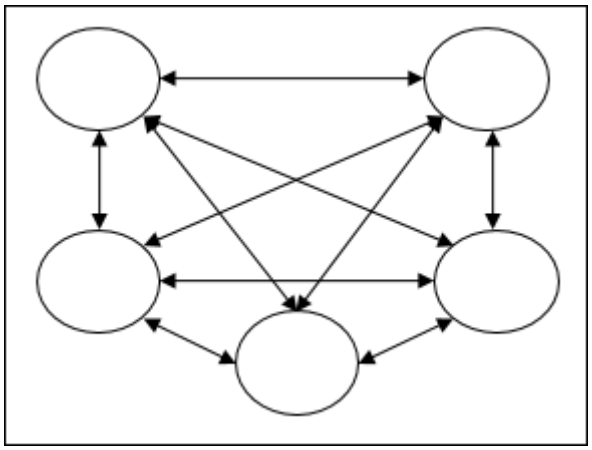

Figure 2.2: A fully recurrent NN. Reprinted from [1].

2.2.2 Learning

The method of modifying the weights of connections between nodes, is defined as *learning*. There are three main categories of machine learning, *supervised learning*, *unsupervised learning* and *reinforcement learning*.

Supervised Learning

This type of learning is dependent, as a NN is trained under supervision. To begin with, the dataset is fixed, adding that every instance is associated with an expected outcome. The general objective is to predict outcomes for new instances. When a NN is trained under supervised learning, its output is compared to the desired outcome, provided by the dataset. Then, based on the difference between the predicted and the actual outcome, the weights are adjusted so as to minimize that difference.

Unsupervised Learning

Unsupervised learning is an independent type of learning, meaning there is no supervision. The dataset is also fixed and the NN learns patterns from unlabeled data. The target outcomes are not provided, so the algorithm learns on its own the structure of inputs. The general objective is extracting information. Such examples are feature extraction, clustering etc.

Reinforcement Learning

A system trained under reinforcement learning, learns incrementally. An agent interacts with a dynamic environment in which the agent performs an action. Then the agent gives feedback, also named *reward*, to the system providing information about the action. The system makes adjustments so as to learn an optimal strategy and maximize the rewards.

2.2.3 Activation Functions

Activation functions in NN are functions which compute the weighted sum of inputs and biases, used for deciding if a neuron can be either fired or not [30]. They are applied over the inputs to get the desired outcome. An activation function can be linear or non-linear, according to the function it represents. In this thesis, the following non-linear functions were used.

Sigmoid Function

Sigmoid function is a non-linear activation function. It is a bounded differentiable real function, defined for real input, adding that it has a positive derivative everywhere [31]. The sigmoid is given by equation 2.1 and is shown in figure 2.3. Due to its shape, sigmoid function is suitable for tasks that require making decisions between two options, such as binary classification.

$$sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Softmax Function

Another activation function is the softmax function [32]. Softmax computes the probability distribution over an output variable of discrete values. In simple terms, it computes the probability of each possible outcome c given an input variable x, i.e. $P(y = c|x)$. Thus, the output ranges from 0 to 1 and the sum of probabilities is equal to 1. The softmax is given by equation 2.2. It is used for multi-class classification tasks and it returns the probabilities of each class. The class with the highest probability is the target.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.2}$$

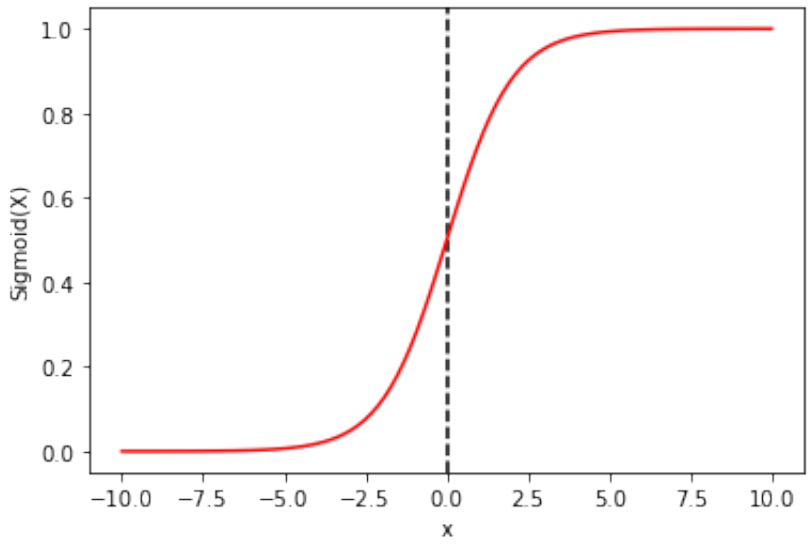

Figure 2.3: Sigmoid function.

Hyperbolic Tangent Function (Tanh)

Hyperbolic tangent function, also known as tanh function, is another non-linear activation function. It is a bounded, zero-centered function and ranges from -1 to 1. Tanh is given by equation 2.3 and is shown in figure 2.4. The tanh function is preferred over sigmoid for training multi-layer networks, as stated in [33].

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

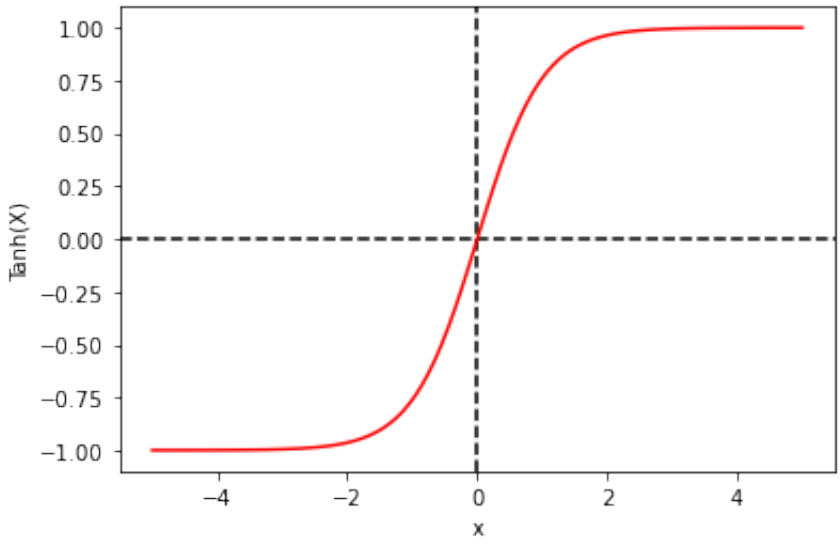

Figure 2.4: Tanh function.

2.3 Gradient-based Training

Training a neural network is the process of minimizing a large-scale , non-convex cost function [34]. Cost functions measure the error between the predicted and the target outcomes. In other words, the purpose of training is to find those parameters of the model that satisfy the best an objective function. This search for the optimal values of the parameters, is named *optimization*. A very common and effective optimization method is the gradient descent. Towards optimization, the entire dataset is passed through several times, updating the values of the parameters. This process is called *epoch*, and the total number of epochs is a hyper-parameter, indicating the total number of times the dataset is going to be passed through.

The steps of a gradient descent training algorithm, for solving a simple linear regression problem $h(x) = b + \sum_{i+1}^{n} \theta_i x_i$, are the following:

1. initialize all parameters θ_i and the bias b of the model to random or heuristic values
2. compute the outcome h
3. compute the cost function $J_\theta(h)$
4. compute the gradients $\frac{\partial J_\theta(h)}{\partial \theta_i}$
5. update all parameters including the bias at the same time, according to the update rule $\theta_i := \theta_i - \alpha \frac{\partial J_\theta(h)}{\partial \theta_i}$, where α is the *learning rate*, see figure 2.5
6. repeat the process until a minimum is reached

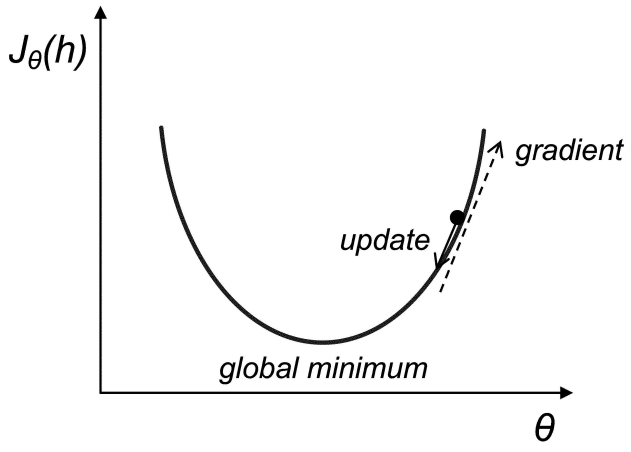

Figure 2.5: Gradient decent. Reprinted from [2].

However, when training a neural network there are more dimensions involved. In that case, the problem has the form $f(x) = b + W^\intercal X$ the gradient of cost function J is defined by equation 2.4, while the updating rule by 2.5.

$$\nabla J(W) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n} \right) \tag{2.4}$$

$$W := W - a \nabla J(W) \tag{2.5}$$

In addition, because of the non-linearity of activation functions, the cost function is not convex. The cost function may have irregularities, hence many local minima instead of an absolute minimum. As a result, gradient descent may stop iterating after reaching a local minimum and not the absolute, as it is supposed to. Saddle points of cost functions are also a major challenge for the gradient descent algorithm [35].

Because of these challenges, more sophisticated methods are usually used as optimizers. Those considered in this thesis are the following.

2.3.1 Stochastic Gradient Descent (SGD)

In stochastic gradient descent, data are shuffled and each sample is considered individually. A sample is drawn randomly, the gradients are calculated with respect to that sample and finally the updating rule 2.5 is applied. The average behavior of the algorithm converges towards the optimal solution, that is the absolute minimum of the cost function.

According to [36], SGD has some benefits over the gradient algorithm. To begin with, SGD converges much faster towards the solution, especially when dataset contains redundant instances, such as duplicates which give identical gradients. As SGD requires only a few instances in each iteration, instead of the entire dataset, it avoids considering many identical instances, hence identical gradients. Moreover, SGD usually does not get stuck in local minima of the cost function, as opposed to the gradient algorithm.

2.3.2 Momentum

An optimizing method, such as SGD, may still be slow. The addition of a momentum term has been shown that accelerates the learning process [37]. Momentum in notated as γ

and its typical value is 0.9. When momentum is used, the updating rule is given by equation 2.6. Signs might be different, depending on the implementation.

$$W := W - a\nabla J(W) + \gamma \Delta W_{t-1} \tag{2.6}$$

2.3.3 RMSprop

RMSprop is an optimizer which is very similar to SGD with momentum. It is an unpublished method proposed by Geoffrey Hinton [38]. In RMSprop , the sum of gradients is defined recursively as a decaying average of past squared gradients [39]. The running average of a timestep is given by equation 2.7, while the updating rule by 2.8.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.7}$$

where $g = \nabla J(W)$.

$$W := W - \frac{\alpha}{\sqrt{E[g^2] + \epsilon}} g \tag{2.8}$$

where ϵ is a smoothing term.

2.4 Long Short-Term Memory

RNN [40] architectures (figure 2.6) are effective with short-term dependencies, but fail when further context is required for predictions. The failure, among other reasons explained in [41], may also be caused by *vanishing gradient*. During training, activation functions' derivatives get multiplied many times. When using functions like sigmoid, derivatives have small values, due to the range of values. As training progresses, the values of derivatives may be so small at some point, that they almost vanish and therefore layers can not be trained properly.

Long Short-Term Memory [42] is a particular type of RNN that can deal successfully with long-term dependencies. Its architecture is depicted in figure 2.7. LSTM consists of *cells*. Cells manipulate network's memory by either remembering or forgetting information, with the assistance of *gates*. There are two kinds of states , the *hidden state* and the *cell state*, that are transferred to the next cell. The architecture (from left to right as shown in figure 2.7) is further explained in the following sections.

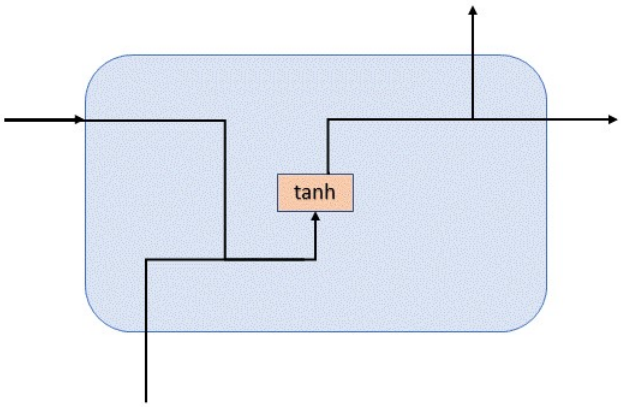

Figure 2.6: Architecture of RNN.

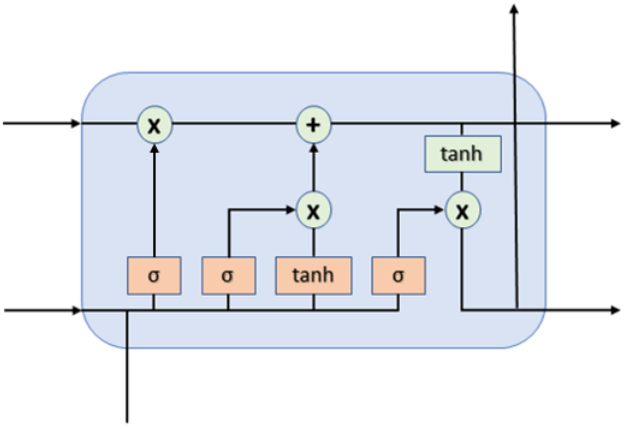

Figure 2.7: Architecture of LSTM. Orange rectangles represent NN layers, while the green circles and rectangle represent pointwise operations.

2.4.1 Forget Gate

Forget gate is the one deciding whether information is kept or not. The previous cell's hidden state or output, as well as the current network's input, are fed to the gate as input. Inputs get multiplied by weights and the bias is added, followed by sigmoid activation. As a result, the output is a vector with values within $\{0, \ldots, 1\}$. The closer to 1, the more important it is to remember this piece of information. The output vector is multiplied to the cell state.

2.4.2 Input Gate

Input gate adds information to the cell state. Firstly, it manages which values to add by filtering them with a sigmoid function, similarly to the forget gate. Then, using tahn function, it creates a vector comprised by all possible values. Values of the output range from -1 to 1. Finally, the two outputs, from sigmoid and tahn functions, are multiplied and added to the cell state.

2.4.3 Output Gate

Principally, the output gate selects useful information from the cell state and outputs it. At first, it scales the values of the cell state with tahn function. With a filter similar to the forget output's, it separates the useful information for the output from the redundant. Lastly, it sends as an output the useful information as well as the hidden state.

2.5 Evaluation of model

There are several metrics to evaluate a model's performance. Those used in this thesis are explained in the following sections.

2.5.1 Cross-entropy

Cross-entropy is a measurement for the error between two probability distributions. It is a common cost function for classification tasks, both binary and multi-class. Its benefit over other cost functions is that it simulates also rare events [43]. The cross-entropy of two discrete probability distributions P and Q with the same support X is given by equation 2.9.

There are two types, the *binary cross-entropy* used for binary classification tasks as well as the *categorical cross-entropy* used for multi-class classification.

$$H(P,Q) = -\sum_{x \in X} P(x) \log Q(x) \tag{2.9}$$

2.5.2 Area under the ROC Curve (AUC)

Area under the ROC Curve is one of the fundamental metrics for evaluating a classification model. ROC is a probability curve and AUC represents the ability of the model to separate the classes [44]. The bigger the value of AUC is, the more capable the model of discerning classes. The ROC curve is depicted in figure 2.8. It is the plot of *true positive rate (TPR)*, also called *recall* or *sensitivity*, against *false positive rate (FPR)*. TPR and FPR are given by equations 2.10 and 2.11, respectively.

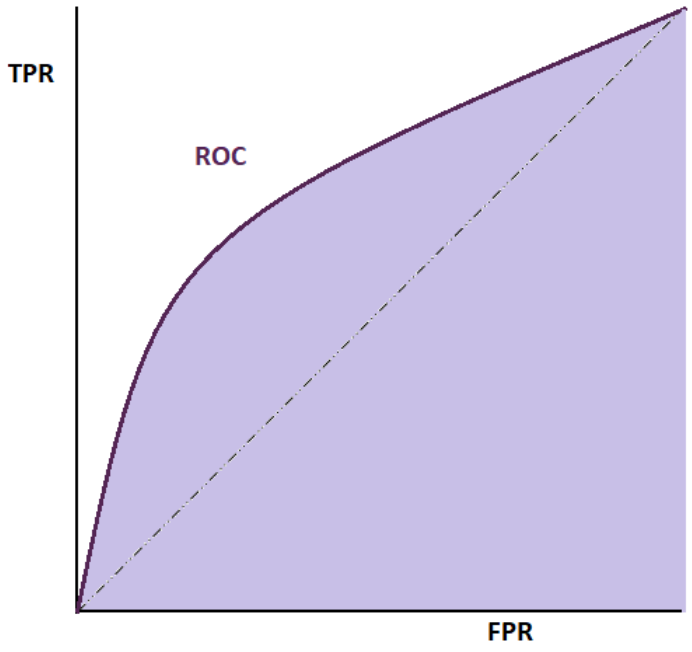

Figure 2.8: The ROC curve.

$$TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.10}$$

$$FPR = 1 - \frac{\text{False Positives}}{\text{True Negatives} + \text{False Positives}} \tag{2.11}$$

Chapter 3

Data

This chapter aims to describe the nature of data used. Furthermore, the encoding of data is analyzed, as well as the pre-processing applied, in order for data to be processed by the model.

3.1 Description of data

It is critical to consider the representation of the musical content, on the grounds that representation and its encoding are firmly associated to the configuration of the input and output of the model. In general, the two different representations that are considered are audio and symbolic. Audio representation of music corresponds to continuous variables, while symbolic to discrete. Even though both ways of representation do not differ much when it comes to processing from a deep learning architecture, the majority of current architectures for music generation choose the symbolic one, as mentioned in [2]. Based on that fact, the symbolic representation of music is, also, considered in this thesis.

More specifically, a data set of MIDI files is used for music generation. MIDI is an acronym for Musical Instrument Digital Interface, which is a technical standard that describes a protocol, a digital interface and connectors and as a result it provides a way of exchanging data between different software applications and devices [45]. This format supports multi-track music, that is music comprised of parallel tracks, each one with distinct sequence of notes. MIDI files consist of event messages that correspond to real-time note performances and control data. Each event is set in a track chunk, which consists of a *delta-time* value and the event. Delta-time specifies the timing an event occurs and it could be either a relative

metrical time or an absolute time. In case of relative metrical time, the number of ticks (measurement of musical time), from the beginning or from the previous event, is specified to indicate the start of an event.

There are many different events [46], but the ones that were used in the process of generation are the following:

- *Note on*, an event that implies a note is played. This event also contains some characteristics. The first one is the *channel number*, which correlates with the instrument that plays the note. The channel number is an integer within $\{0, 1, \ldots, 15\}$. Moreover, a *pitch number* is contained, a integer within $\{0, 1, \ldots, 127\}$ that indicates the note pitch. Lastly, the *velocity* is designated by an integer within $\{0, 1, \ldots, 127\}$, and implies the loudness of the note played.

- *Note off*, an event that indicates a note is no longer played. The Note off event has the same characteristics as the Note on, however velocity specifies how fast a note is released.

- *Set Tempo*, an event, usually in the beginning of each track, that specifies its tempo. In case there is not a Set Tempo event, the default value is 120 beats per minute (BPM).

- *Program Change*, an event that tells a device a certain program should be selected for a MIDI channel. At most times this means that a certain instrument is selected for the channel, therefore the following notes will be played by this instrument.

- *End Of Track*, an event that indicates when a track ends.

3.2 Track recognition

For music generation, especially when dealing with multi-track music, it is fundamental to specify the type of each track. To elaborate further, melody track is very important in music generation but commonly it is not played by a single instrument and consequently a single MIDI channel. Thus, it is required to recognize the melody track, as well as other types, such as bass and drum tracks, for generating accompaniment music.

MIDI Miner [47] was used in PopMAG [26] for track type recognition. MIDI Miner is a Python library that implements a random forest classifier, which recognizes melody, bass,

chord, drum and accompaniment tracks. This library was also employed for the purpose of this thesis.

3.3 Encoding of data

Once the format of data is determined, data have to be encoded in order to get processed by any model. Having discrete variables and 128 different pitch values in MIDI files, every channel track can be encoded into a two dimensional array. The array consists of 128 rows, each one corresponding to a pitch value and as many columns as the total ticks of the track, each one representing a single tick. The values of the cells are the velocity values of Note on and Note off events, with positive and negative sign respectively. The velocity of each pitch is repeated in the array, for as many columns as the total ticks the pitch is played. As a result, the array created is very similar to MIDI file visualization, see figures 3.1 and 3.2.

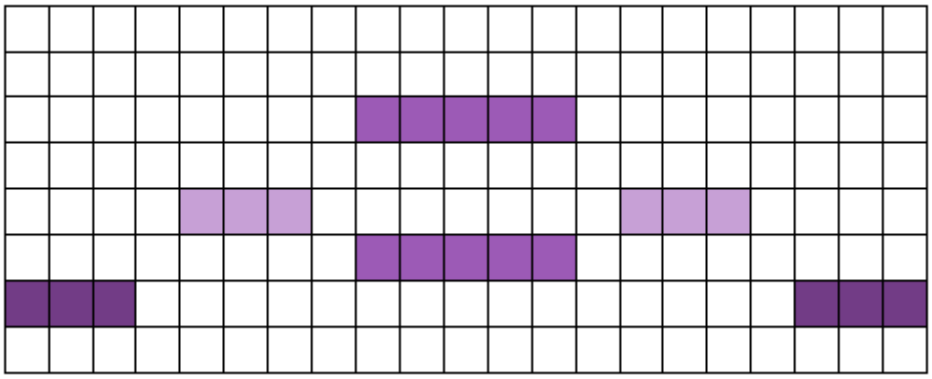

Figure 3.1: A visualization of MIDI file. Each column corresponds to one tick, while each row to a pitch (in this figure only 8 out of 128 pitches are shown). The darker the colour of the bar, the bigger the velocity.

To conclude, each MIDI file is encoded in two dimensional arrays, corresponding to types of tracks recognized by MIDI Miner. Python-midi library [48] was used in order to easily access MIDI files' content, as well as write MIDI files.

3.4 Data binning

In music generation, it is important to consider dynamics, that is the variation in loudness. Dynamics add expressiveness to a music piece, otherwise it sounds too standardized. The two

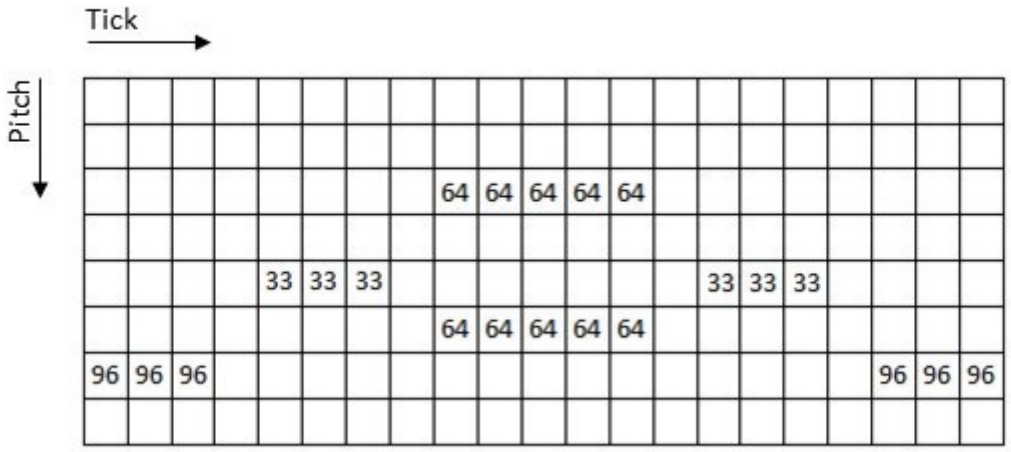

Figure 3.2: The array corresponding to the MIDI visualized in figure 3.1.

basic dynamic indicators are ***f*** and ***p*** meaning ”loud” and ”quiet” respectively [49].

In MIDI files dynamics are represented by the velocities of Note on and Note off events. There are 128 total distinct velocity values and since music pieces are unique, it would be challenging for any model to learn music patterns. Most of the systems that have been developed so far, usually consider a binary representation of note events, for example 0 meaning Note off and 1 Note on. Such representation would lead to composing music lacking dynamics, hence lacking expressiveness. On the other hand, dealing with the entire range of velocity is a challenge. Therefore, velocity values were binned.

Data binning or *bucketing* is a common feature engineering technique used not only to transform continuous variables into discrete, but also to reduce the cardinality of discrete values [50]. Values are divided into bins and each bin gets replaced by a representative value, usually the center (e.g. the mean or the median). The values of velocity were binned so as to match the scale of dynamics. The representative values, listed in table 3.1, are the same as the defaults for MuseScore [51], a music composition and notation software.

So values were binned according to the following rules:

- $v = 0$ (remains the same)
- $0 < v \leq 16 \implies v = 16$
- $16 < v \leq 33 \implies v = 33$

Indicator	Level	Velocity
fff	very very loud	127
ff	very loud	112
f	loud	96
mf	average	80
mp	average	64
p	quiet	49
pp	very quiet	33
ppp	very very quiet	16

Table 3.1: Velocity values associated with dynamics indicators for MuseScore.

- $33 < v \leq 49 \implies v = 49$
- $49 < v \leq 64 \implies v = 64$
- $64 < v \leq 80 \implies v = 80$
- $80 < v \leq 96 \implies v = 96$
- $96 < v \leq 112 \implies v = 112$
- $112 < v \leq 127 \implies v = 127$

where v is the velocity value of a MIDI event.

Chapter 4

Model Building

The aim of this chapter is to analyze the NN architectures built, in order to generate music content.

4.1 Objective

To begin with, the general objective of the model needs to be specified. The objective may include for example if the goal is to generate content ex nihilo (out of nothing) or generate accompaniment content. Consequently, the model is being built with regard to its goals.

In this thesis, the objective consists of the following goals:

- both ex nihilo and accompaniment generation
- length variability
- expressiveness
- melody-harmony consistency

Ex nihilo generation was set as goal so as to create original melodies, provided only a *seed* (input vector). In addition, generating accompaniment music content for the melody is desired, in order to create multi-track music content. In other words, the aim is to generate melody as well as support to the melody.

Length variability was also desired for melody generation. This means that the total number of ticks will not be fixed, hence the duration of the content generated may vary. Duration

of each piece is set by the user. Length variability applies only to melody generation, as the accompaniment music's duration equals melody's duration.

Expressiveness is an important factor in music, thus dynamics were considered, as already mentioned in chapter 3.

Last but not least, the generated content is wanted to be melody-harmony consistent. Harmony is defined as the combination of simultaneously played notes, also known as chords, while melody is a sequence of single notes. In order for a music piece to sound pleasing, melody and harmony should interact. By examining music from the encoding's perspective (described in section 3.3), melody is associated with the horizontal axis (tick dimension), while harmony with the vertical axis (pitch dimension). Consequently, the model must consider sequences in both axes.

Two separate models were built for satisfying the objective. A *melody model* for generating the melody track, as the name suggests, and an *accompaniment model* for generating support content in respect to melody. They both share the same architecture, however they are configured differently towards achieving different goals.

4.2 Architecture

The architecture is mainly inspired by Johnson's proposed biaxial LSTMs [17]. This model also considers sequences in both axes of data, adding that it employs Bidirectional LSTM instead of LSTM layers, as it is found to perform better [21]. The architecture, described in the following sections, is originally built for the purpose of this thesis.

The NN built consists of two separate, pre-trained subnetworks merged into one. The first subnetwork, *pitch model*, is responsible for modelling the chords, while the second, *time model*, for temporal sequences and dynamics of the notes.

All networks employ LSTM layers, described in section 2.4, towards modelling the probability distributions, with Bidirectional [52] wrapper layer. The Bidirectional wrapper combines two hidden RNN layers, in this case LSTM layers, of opposite directions. Therefore, the networks are able to model more effectively long-term dependencies of music content.

For the implementation of the NN, Keras [53], an open-source software library with Python interface, was employed. For visualizing the architectures Netron [54], a visualizer for DL and ML models, was used.

4.2.1 Pitch Model

The pitch model learns patterns in the vertical dimension of data, along the pitch axis (see figure 3.2). This way, the network models the probabilities of notes occurring simultaneously, given the previous chord played. This model is the only one that considers binary representation of music content, so as to focus only on chords learning and not dynamics. Namely, the encoded array contains ones, in case a note is being played, and zeros otherwise.

A single tick of shape $(128, 1)$ is fed into the model and the next tick is output. The input's shape indicates that samples consist of 128 timesteps, corresponding to 128 pitches, and one feature, as it is fed one tick at a time (see figure 4.1). As follows, the model learns which notes are more likely to occur, based on previous pitches' values.

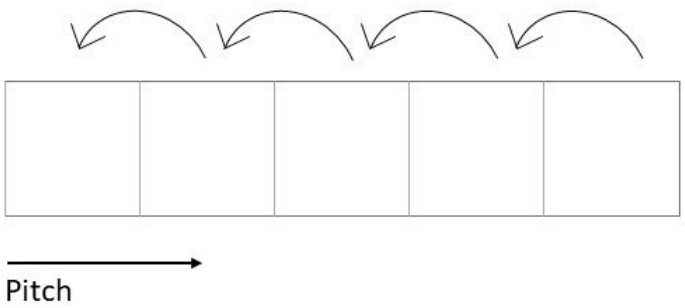

Figure 4.1: The pitch model's input. The arrows indicate timesteps of sequences. In this figure only 5 out of 128 pitches are displayed.

The network is comprised of 6 total layers, apart from the input layer:

1. Bidirectional LSTM layer. The LSTM hidden layers have 128 units and tanh activation function. They also return the sequences, meaning that they return all the timesteps of input.

2. Dropout layer with 0.3 rate. The Dropout layer sets randomly some units to 0, during training, to prevent overfitting.

3. Bidirectional LSTM layer. Each LSTM layer consists of 128 units, tanh activation function and returns sequences.

4. Bidirectional LSTM layer. Each LSTM layer has 64 units, tanh activation function and returns sequences.

5. Dense layer with Time Distributed wrapper, with 1 unit and sigmoid activation function. The Time Distributed wrapper layer basically applies sigmoid activation to every timestep of the input, hence to all pitches.

6. Flatten layer, in order to map the output to a single column vector.

The number of layers and their parameters were determined through experiments, with the intention of achieving accuracy scores as high as possible. Its architecture is depicted in figure 4.2.

4.2.2 Time Model

The time model is responsible for learning patterns in the horizontal dimension, along the tick axis (see figure 3.2). Thereupon, the network models the probabilities of notes occurring directly after a provided sequence of notes. Dynamics are considered, as a result the network's output contains also the probabilities for each bin of dynamics.

More specifically, the model is fed a sequence of 240 ticks (see figure 4.3) and predicts the next tick. Therefore, the model is trained to predict which notes are more likely to occur after a given sequence, adding the loudness of those notes.

The model is made up by 7 total layers, apart from the input layer:

1. Bidirectional LSTM layer. Each LSTM hidden layer has 128 units, tanh activation function and returns sequences.

2. Dropout layer with a rate of 0.3.

3. Bidirectional LSTM layer. The LSTM layers consist of 64 units, have tanh activation function and return sequences.

4. Dropout layer with 0.3 rate.

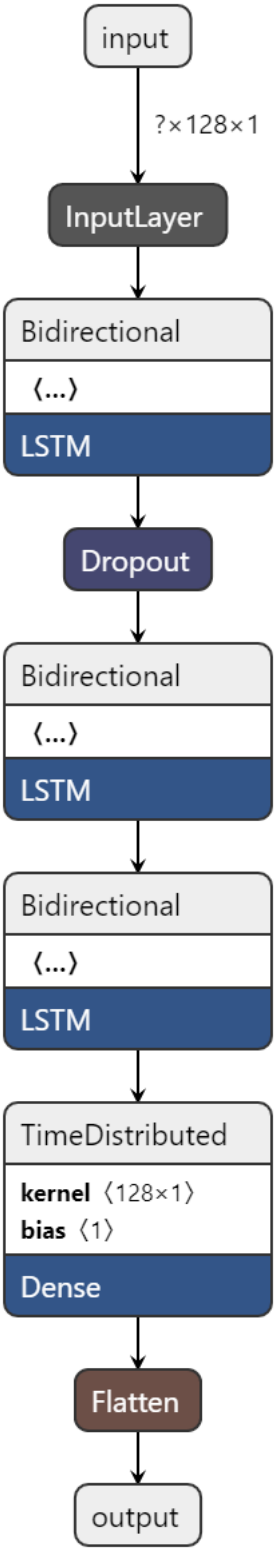

Figure 4.2: Pitch model's architecture.

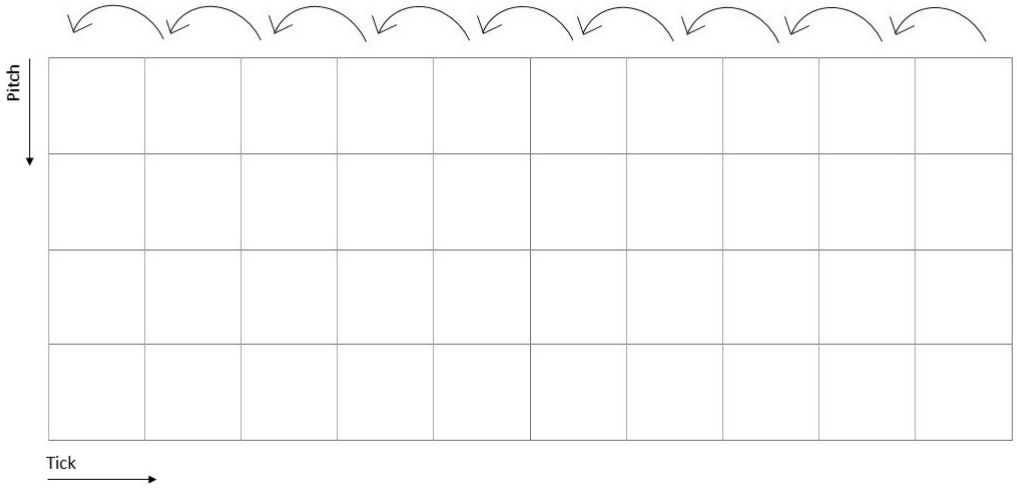

Figure 4.3: The time model's input. The arrows indicate timesteps of sequences. Only 10 out of 240 ticks and 4 out of 128 pitches are displayed in this figure.

5. Bidirectional LSTM layer. Each hidden layer is comprised of 64 units, tanh activation and returns sequences.

6. Permute layer to rearrange the dimensions of the vector. The previous layer's output has shape (240,128). Towards applying the final activation along the pitch axis, the vector needs to be transposed.

7. Dense layer with 9 units (as the total number of possible dynamics' values) and softmax activation. As a result, the final output has shape (128,9), which means that to each pitch value correspond 9 probabilities, one for each dynamic bin.

The total number of layers and their parameters were specified through experiments, towards achieving accuracy scores as high as possible. Its architecture is shown in figure 4.4

4.2.3 Merged Model

The models described in sections 4.2.1 and 4.2.2 are merged into one network, in order to make predictions. Both networks are trained to model probability distributions of notes,

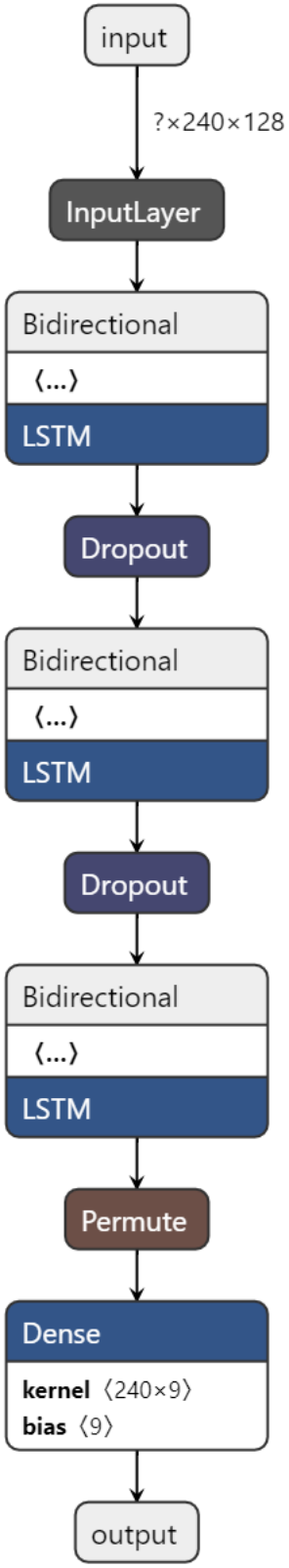

Figure 4.4: Time model's architecture.

considering sequences in distinct axes of data. For the purpose of making predictions based on both distributions, the merged network multiplies their outputs. Thus, the probability of a note occurring is calculated not only by the pitch model, but also from the time one.

For the two outputs to be multiplied element-wise, their shapes have to match. So, the pitch model's output (a single column vector of shape 128) is repeated for 9 total columns, resulting in an array with shape $(128, 9)$, after a rearrangement of dimensions.

The merged model has two additional Dense layers with Time Distributed wrapper, towards achieving higher accuracy and AUC score. The first layer is comprised of 256 units with tanh activation, while the second of 9 units (as the range of dynamics) and softmax activation. Hence, the final output is an array with shape $(128, 9)$ containing the probabilities of dynamics corresponding to each pitch.

Its architecture is displayed in figure 4.5.

4.3 Melody Model

The melody model uses sequences of notes only from the melody track. In addition, pauses were removed from the encoded melody track. In other words, ticks that contain no notes were not considered by the melody model, on the grounds that AUC score was significantly higher and the generated content was more pleasing, compared to having pauses considered.

4.3.1 Pitch Model

The pitch model is fed with the last tick played as input, and it then predicts the probabilities of the next tick's notes. It is compiled with RMSprop optimizer, an initial learning rate of 0.001 and momentum of 0.9. Binary cross-entropy is used as a loss function, since the model considers binary representation of notes. The model is trained for 100 epochs with a batch size of 100 and a learning rate scheduler. The learning rate scheduler adjusts the learning rate during training. For this task, the network achieved higher accuracy scores with a decaying rate. Thus, the learning rate is set to decay according to the 4.1 rule. Basically, the rate is decreased by half every 10 epochs.

$$\text{learning rate} := \text{learning rate} \cdot 0.5^{\lfloor \frac{1+epoch}{10} \rfloor} \tag{4.1}$$

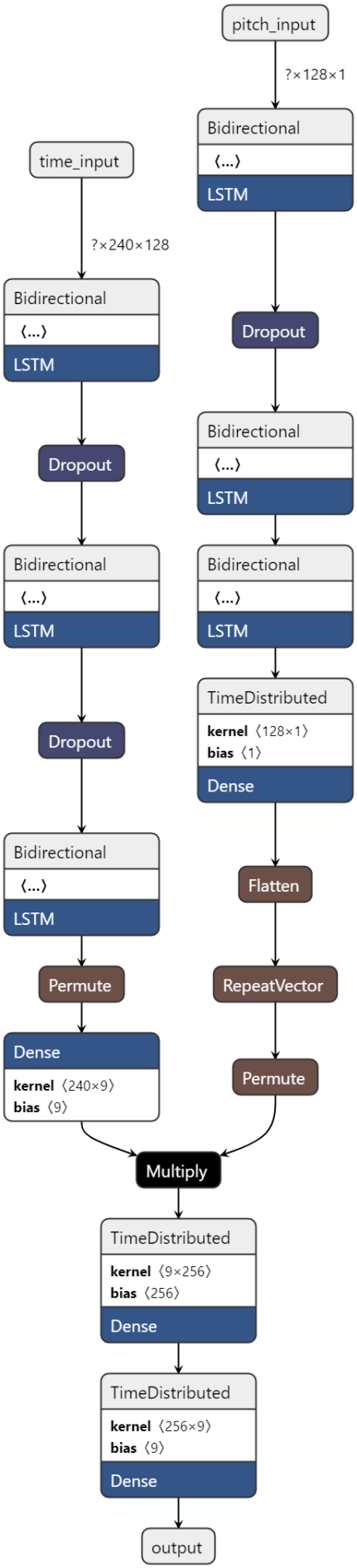

Figure 4.5: Merged model's architecture.

4.3.2 Time Model

The time model is fed a sequence of 240 successive ticks to predict the next tick. The length of sequence was chosen through experimentation, in order to capture changes in melody. The model is compiled also with RMSprop optimizer, an initial learning rate of 0.001 and momentum of 0.9. Categorical cross-entropy was used as loss function, on the grounds that the outputs are one-hot encoded. It was trained for 50 epochs and a batch size of 100. Time model, also, adjusts the learning rate according to rule 4.1.

4.3.3 Merged Model

The two pre-trained models are merged into one. Consequently, the final model is fed the inputs of pitch and time networks. In figure 4.6 the inputs are displayed. The red rectangle corresponds to time model's input (only 13 ticks are shown instead of 240), while the blue rectangle is the pitch model's input. The yellow rectangle is the tick to be predicted.

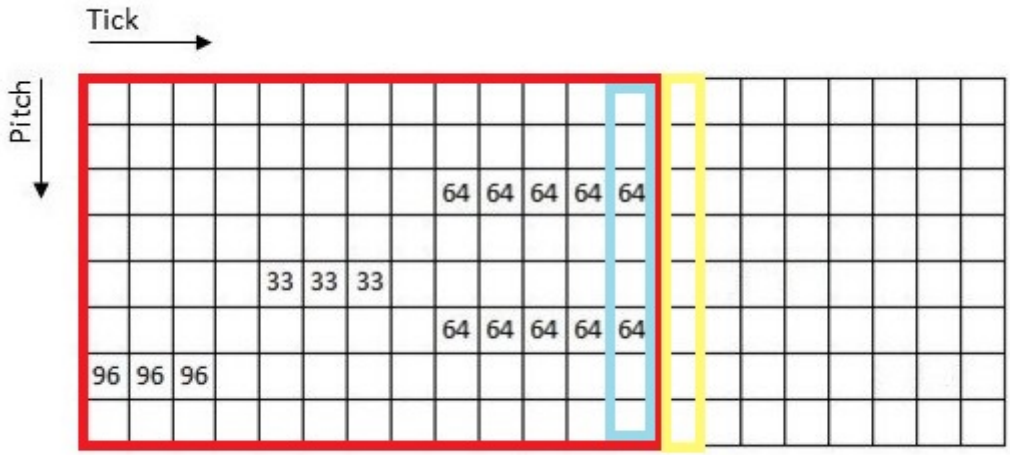

Figure 4.6: A visualization of melody model's inputs. Only 8 pitches are displayed instead of 128.

The merged model is compiled with RMSprop optimizer, learning rate of 0.001 and momentum of 0.9. It is trained for a single epoch with a batch size of 500, so as to minimize categorical cross-entropy.

4.4 Accompaniment Model

The accompaniment model uses sequences from the contextual accompaniment track (e.g. drum track, bass track e.t.c.), as well as the melody track of each song. This is necessary in order for the accompaniment track to interact with the melody one, otherwise the outcome would not be harmonic.

4.4.1 Pitch Model

Accompaniment's pitch model is similar to the melody's pitch model, however the inputs differ. Towards accomplishing interaction between the two tracks, the pitch model is fed a tick from the melody track and it outputs the notes being played in the accompaniment track, at the same tick (time). The network considers also pauses in tracks, otherwise the synchronization between tracks would be lost, however in case both ticks contain no notes they are removed. Therefore, the model predicts which notes are most likely to occur simultaneously with the melody's notes. This model is compiled with the same parameters as the melody's pitch model (see section 4.3.1) and it is trained for 50 epochs with a batch size of 50.

4.4.2 Time Model

Time model processes sequences of 240 ticks from the accompaniment track to predict the next tick, along with dynamics. This way, the model learns patterns of successive notes in the contextual track, independently of the other tracks. Pauses are removed from the encoded tracks. It is compiled and trained with the same parameters as the melody's time model (see section 4.3.2).

4.4.3 Merged Model

The merged model is fed inputs of both pre-trained models. In figure 4.7 the inputs of the accompaniment model are shown. The red rectangle is the input of time model (13 instead of 240 ticks are depicted), the blue rectangle stands for the pitch model's input, while the yellow is the tick to predicted. As shown, pitch model is fed with the tick from the melody track that is occurring simultaneously with the tick from the accompaniment track that is to be predicted. This way, the model is able to accomplish harmony between the two distinct tracks. Because

of the importance of synchronization, pauses are necessary also for this model. However, simultaneous ticks containing no notes are once again removed.

The network is compiled with RMSprop optimizer, learning rate of 0.001 and momentum of 0.9. It is trained for a single epoch, with a batch size of 1000 and categorical cross-entropy as loss function to be minimized.

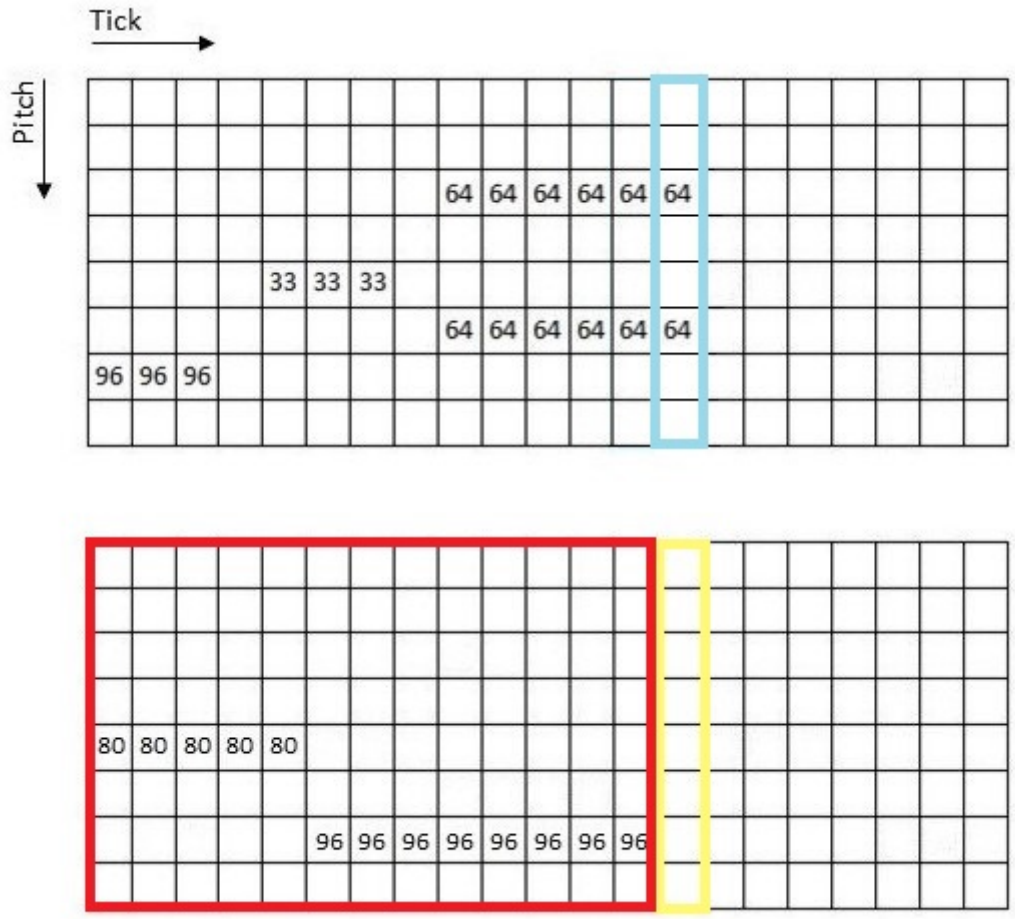

Figure 4.7: Accompaniment model's inputs. The first array stands for the melody track and the second to an accompaniment track of the same music piece. Only 8 out of 128 total pitches are displayed in both tracks.

4.5 Generation

First of all, when a new music piece is to be generated, its tempo (BPM) and duration need to be specified by the user. The duration is given in seconds and it is converted to MIDI

ticks. The actual duration in seconds of a music tick, depends on tempo and resolution of the MIDI track and it is calculated by the following formula:

$$\text{tick} = \lceil \frac{1}{\text{ratio}} \cdot \text{seconds} \rceil \tag{4.2}$$

where ratio is given by

$$\text{ratio} = \frac{60}{\text{tempo} \cdot \text{resolution}} \tag{4.3}$$

The value of resolution is set to the default, which is 220 Pulses per Quarter note [48].

A seed is also necessary for the generation process. The user gives the 240 first ticks of the tracks (melody, bass, drum e.t.c.) of any music piece. Those ticks are binned appropriately and are fed to the models. Firstly, the next tick of the melody track is predicted is predicted, followed by the first tick prediction - in respect to the melody tick - for the rest tracks. The process is continued iteratively, until the desired number of ticks is reached.

An empty array of the prosper shape (#number of tracks, 128, #total ticks) gets initialized. Since both models output arrays of shape (128,9) containing probabilities for each velocity bin, the bin with the highest probability will be chosen for each pitch. As a result, the final tick has shape (128,1) and is added in the array. However, if the same tick occurs more than 300 consecutive times, then the bin with the second highest probability will be chosen for the less likely pitch to occur. To put it in simple words, the pitch that is less likely to occur compared, will be played. That way, models are prevented from predicting the same notes over and over again.

By the end of the process, the array is decoded in a MIDI file accordingly to the encoding process explained in section 3.3.

Chapter 5

Results

The aim of this chapter is to present the results of the models and evaluate them.

5.1 AUC scores

The models were tested for four different music genres:

- classical
- blues
- rock
- jazz

Therefore, each model was trained using four distinct datasets containing MIDI files from the different genres. For generating classical music, Classical Music MIDI [55] dataset was employed, which contains the melodies of 295 pieces. Blues Genre MIDI Melodies [56], which contains the melody tracks of 122 pieces, was used for generating blues melodies. For the generation of rock and jazz content, the MIDI files used were collected manually from the Internet.

Classical, blues and rock datasets contain only melody tracks, hence only the melody model was trained. Jazz dataset has polyphonic content, meaning that the accompaniment model was also trained for each track.

The losses and AUC scores during the training phase are presented in the following sections.

5.1.1 Classical genre

The training history of the pitch model is depicted in figure 5.1. As shown in the graphs, the pitch model achieved an AUC score of 0.99, hence performed well and managed to model accurately the harmony of the classical content.

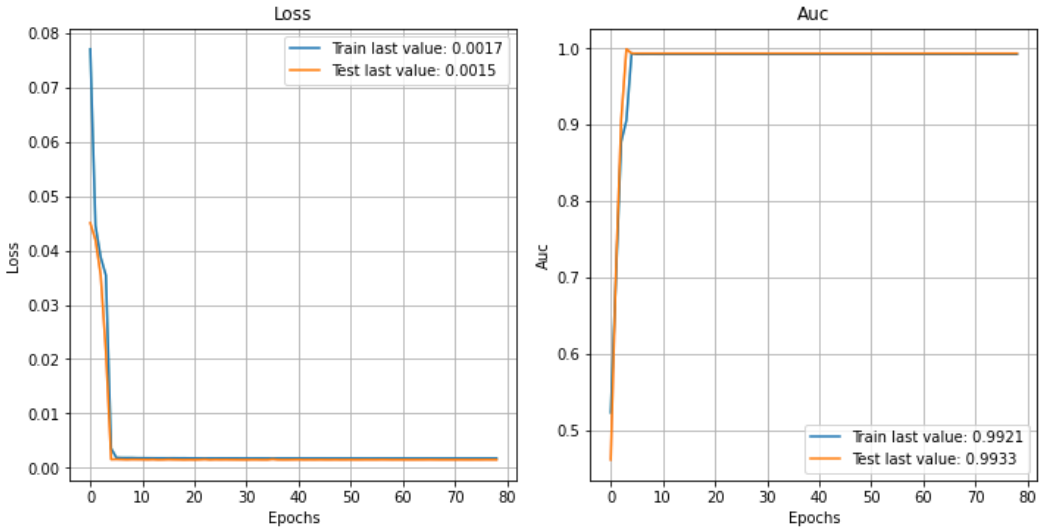

Figure 5.1: Training history of classical pitch model.

The training history of the time model is displayed in figure 5.2. Time model also performed well, achieving 0.99 AUC score, thus the temporal sequences along with dynamics were modelled accurately.

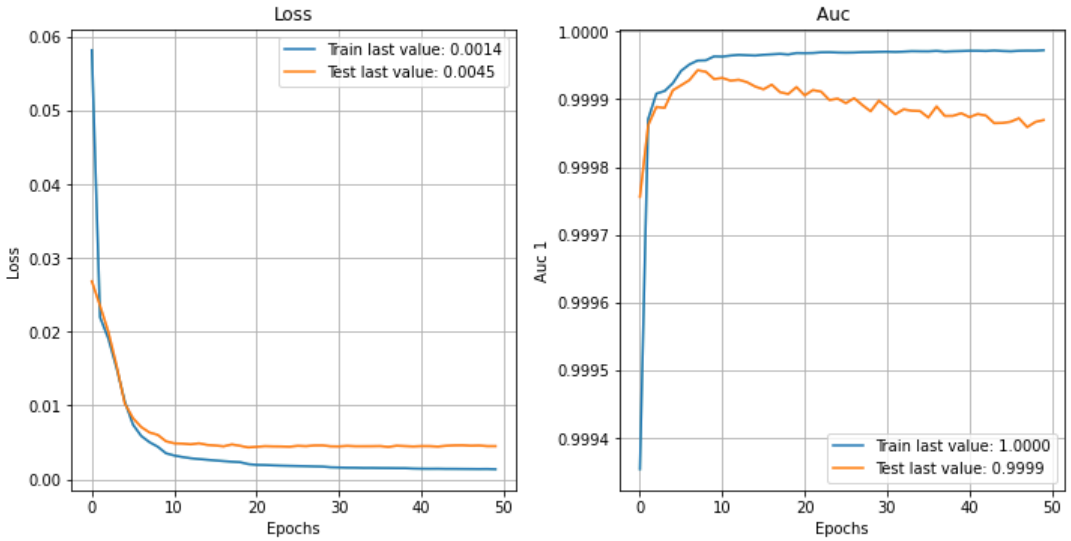

Figure 5.2: Training history of classical time model.

Since the merged melody model is trained for a single epoch, the training history can not

be depicted. The last value of the loss function was 0.0072, while the final AUC score was 1. This indicates that the network managed to model accurately the underlying structure of the classical content.

5.1.2 Blues genre

The pitch model's training history is depicted in figure 5.3. The model achieved 0.99 AUC score, indicating that the harmony was modelled with high accuracy.

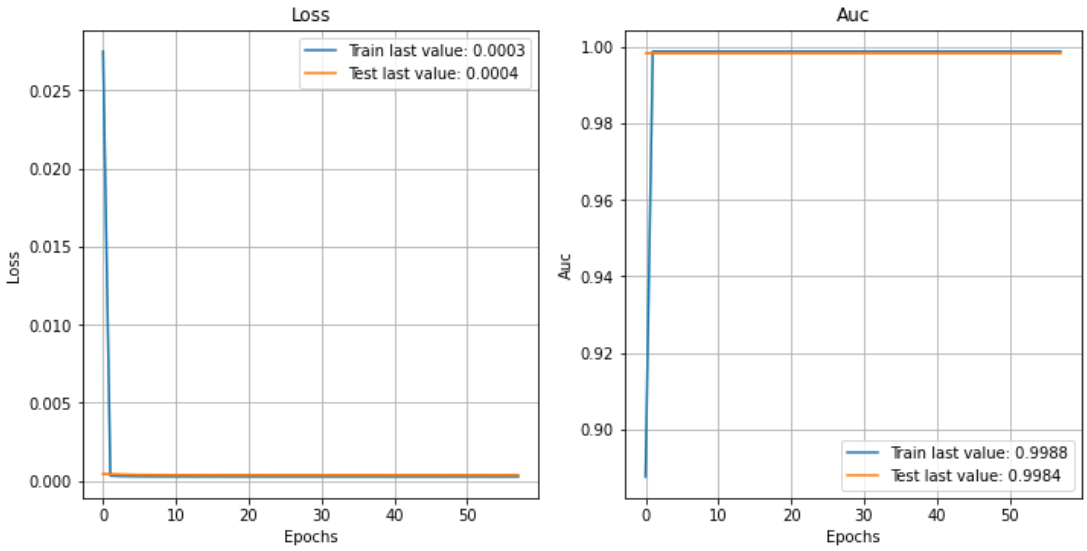

Figure 5.3: Training history of blues pitch model.

In figure 5.4, the training history of the time model is displayed. As shown in graphs, the time model achieved an AUC score 0.99, meaning that the temporal sequences and dynamics of the notes were captivated very accurately.

The final merged model achieved an AUC score of 1, while the last value of the loss function was 0.015. As a result, the network performed well on the blues dataset and modelled its musical structure.

5.1.3 Rock genre

The training history of the pitch network is displayed in figure 5.5. The model achieved an AUC score of 0.97, hence the harmony of the rock dataset was modelled accurately enough.

The time model's training history is shown in figure 5.6. The network achieved 0.99 AUC score, therefore the successive notes along with their dynamics were captured with precision.

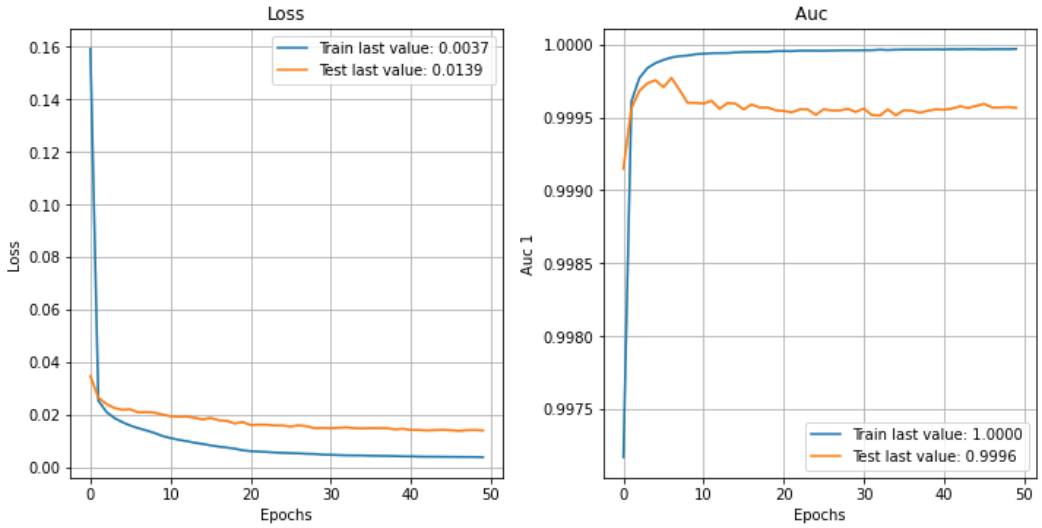

Figure 5.4: Training history of blues time model.

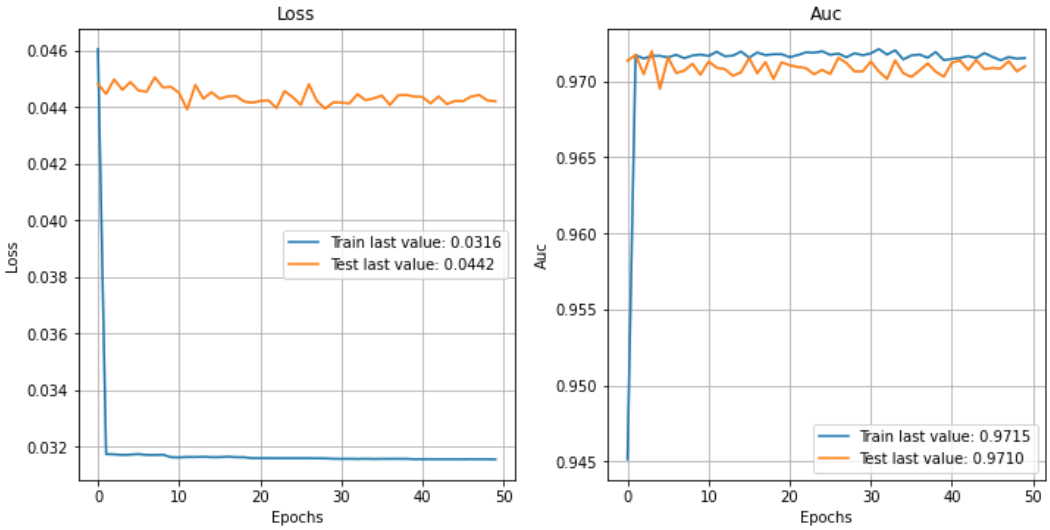

Figure 5.5: Training history of rock pitch model.

The merged model's final AUC score was 0.99, while the final value of the loss function was 0.0124. So overall the model seems to have modelled successfully rock melodies.

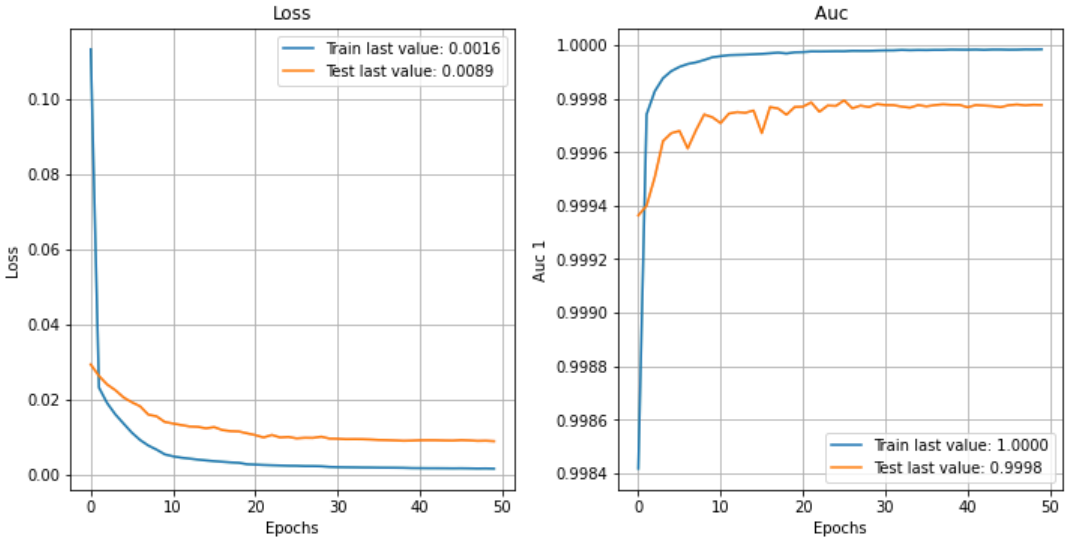

Figure 5.6: Training history of rock time model.

5.1.4 Jazz genre

The jazz dataset contains multi-track pieces, so both models were trained on this one. The melody model was trained on melody tracks, while the accompaniment model was trained on the bass, drum and accompaniment tracks.

Melody track

The pitch model's training history is depicted in figure 5.8. As shown, the pitch model achieved a high AUC score of 0.99, meaning that the harmony of the melody tracks was captured accurately.

The time model's training history is displayed in figure 5.8. This network, also performed well, reaching an AUC score of 0.99. Correspondingly, successive notes and their dynamics were modelled successfully.

The final merged model achieved an AUC score of 1, adding that the last value of the loss function was 0.0183. So overall the network seems to have modelled efficiently the structure of the melody tracks.

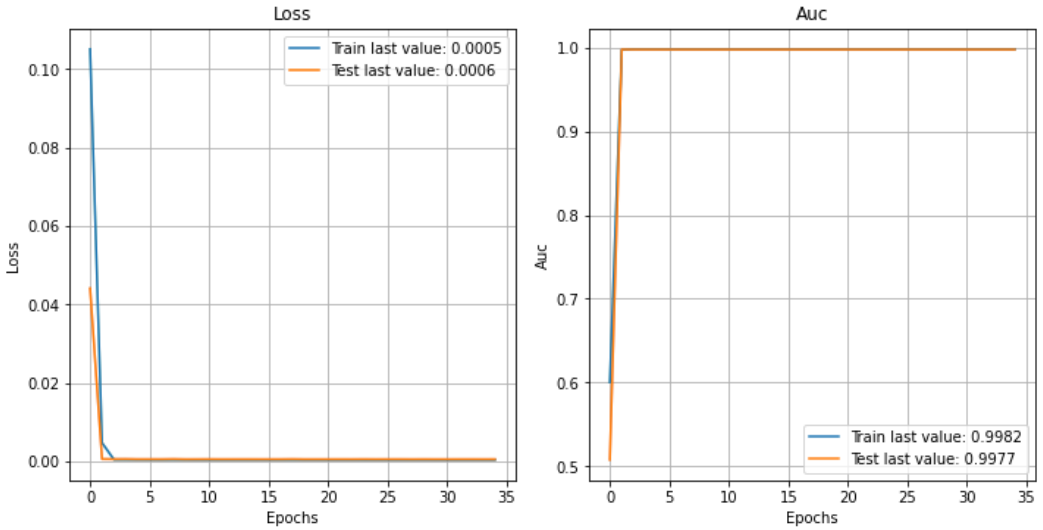

Figure 5.7: Training history of jazz pitch model for the melody tracks.

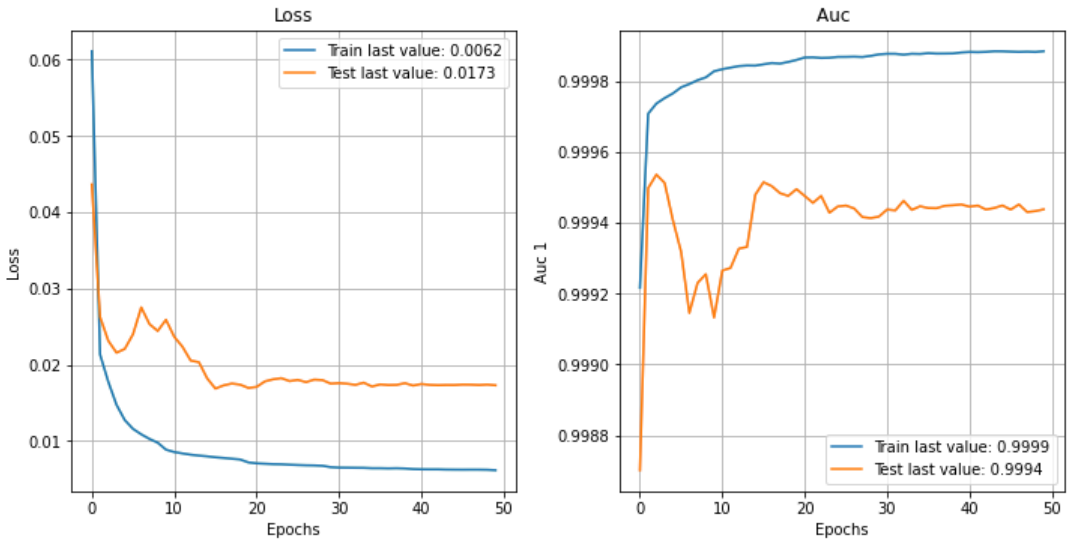

Figure 5.8: Training history of jazz time model for the melody tracks.

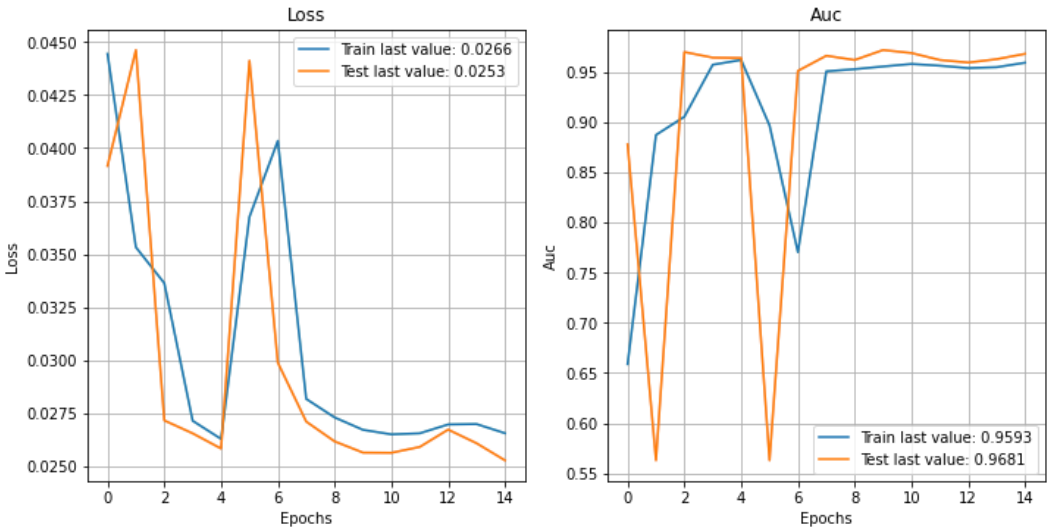

Figure 5.9: Training history of jazz pitch model for the bass tracks.

Bass track

In figure 5.9 the training history of the accompaniment's pitch model is shown, when trained on the bass tracks. The model reached 0.96 AUC score, indicating that the harmonic structure, between the melody and the bass tracks, was accurately modelled.

The time model's training history is displayed in figure 5.10. The network achieved an AUC score of 0.99, as a result the temporal sequences along with the notes' dynamics were modelled accurately.

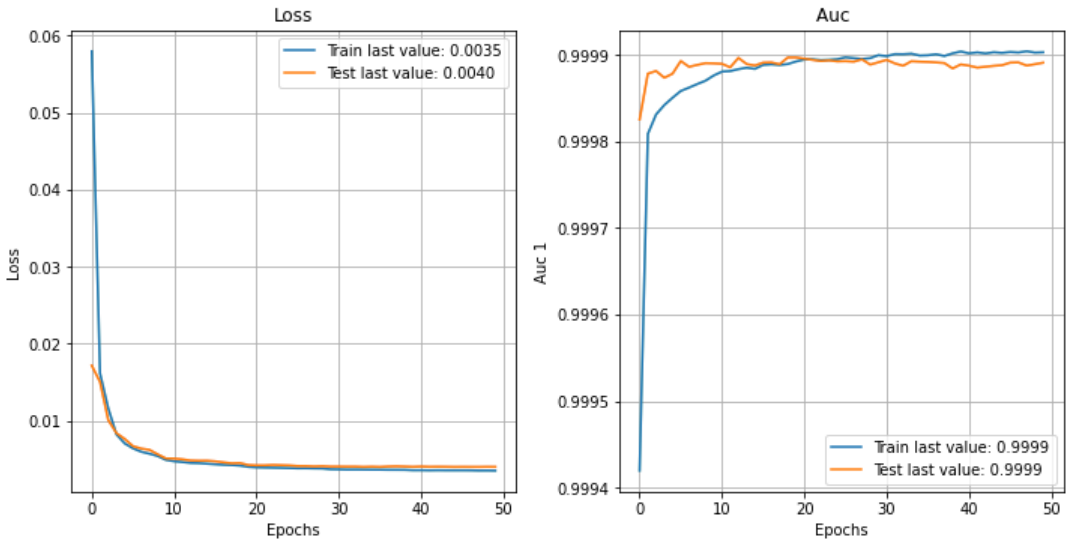

Figure 5.10: Training history of jazz time model for the bass tracks.

The merged model's final AUC score was 0.99, while the last value of the loss function

was 0.0123. So the accompaniment network managed to model the structure of the bass tracks in respect to the melodies.

Drum track

The pitch model's history regarding the drum tracks is depicted in figure 5.11. The harmony between melody and drums was captured with high accuracy, as the model achieved an AUC score of 0.99.

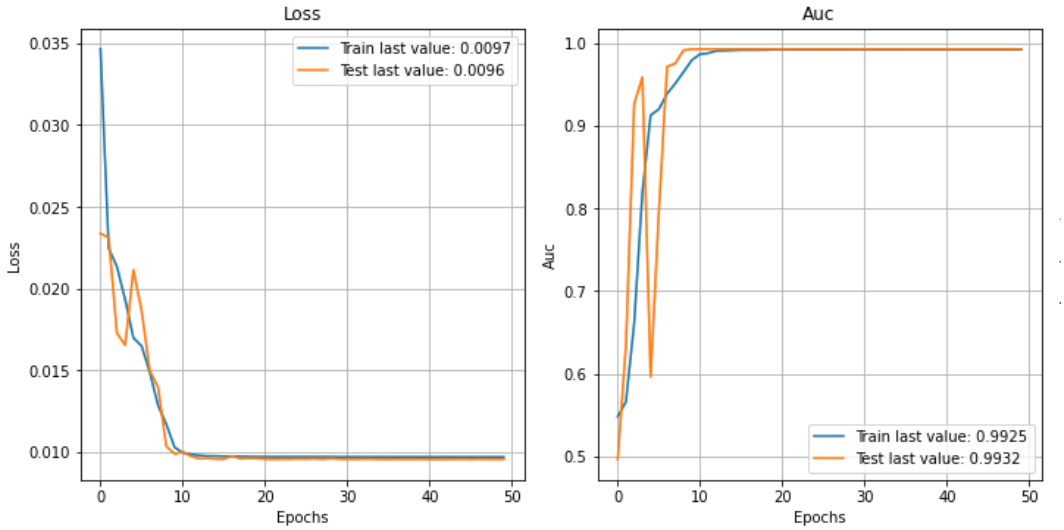

Figure 5.11: Training history of jazz pitch model for the drum tracks.

In figure 5.12, the training history of time model, for the drum tracks, is displayed. This network also achieved 0.99 AUC score, meaning that successive notes and their dynamics were modelled with precision.

The merged model achieved also 0.99 AUC score, adding that the final value of the loss function was 0.0143. Therefore, drum tracks were accurately learned by the accompaniment model.

Accompaniment track

The training history of the pitch model, regarding the accompaniment tracks, is shown in figure 5.13. Even though the model scored high AUC for the training set and low values of the loss function, the network did not manage to predict the chords accurately.

In figure 5.14 the training history of the time model is depicted. The model achieved an AUC score of 0.99,meaning that temporal sequences along with the notes' dynamics were

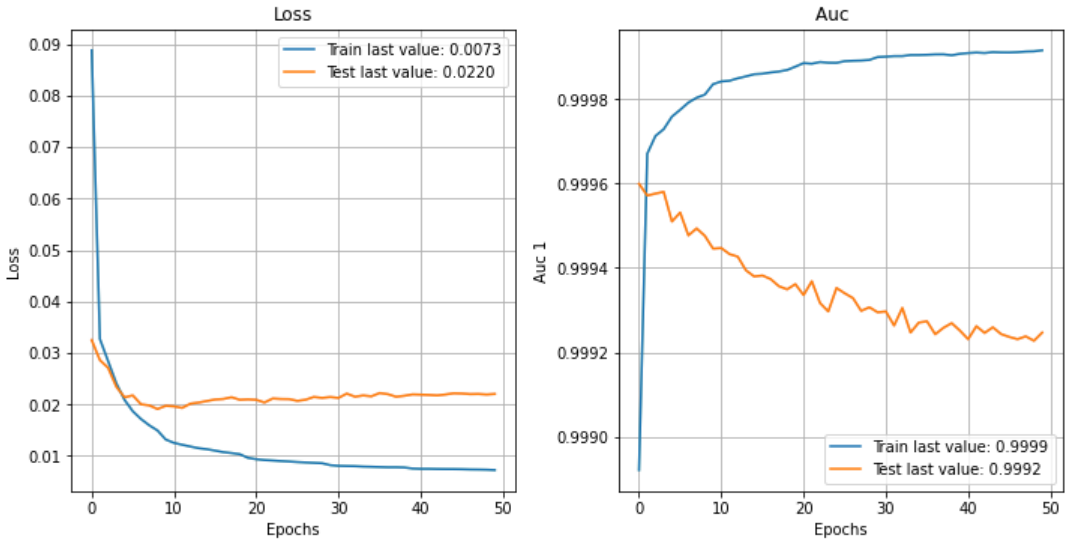

Figure 5.12: Training history of jazz time model for the drum tracks.

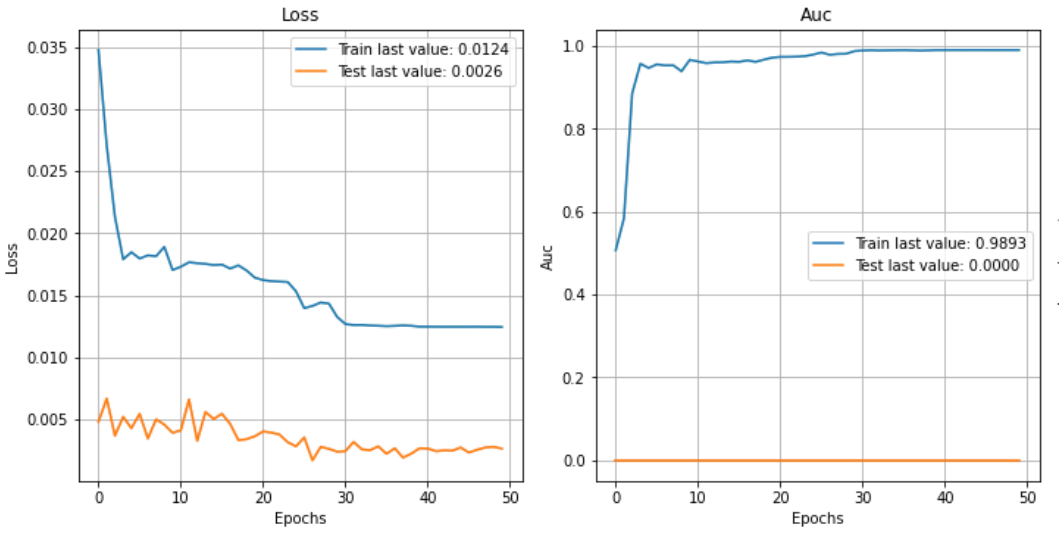

Figure 5.13: Training history of jazz pitch model for the accompaniment tracks.

modelled accurately.

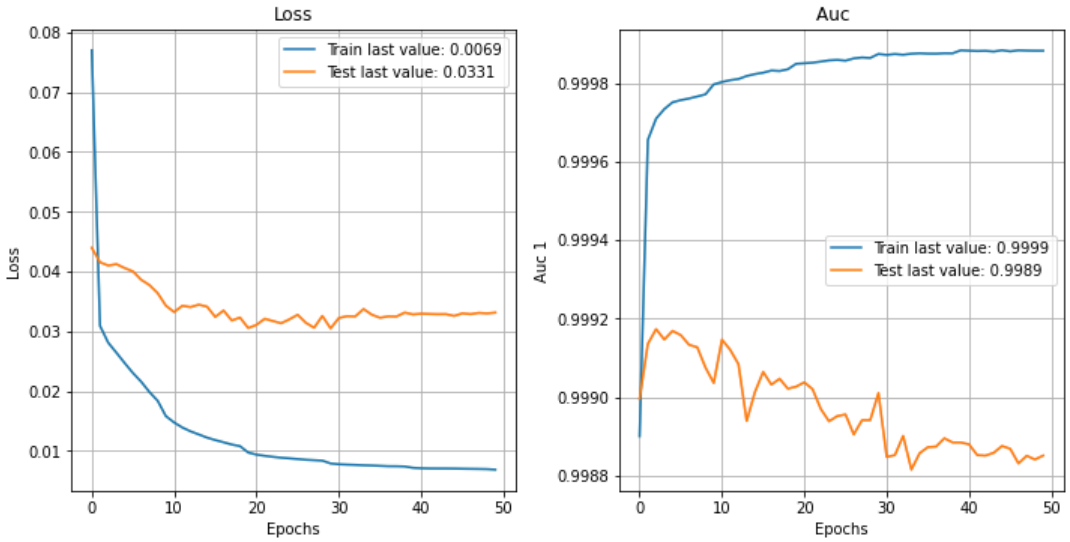

Figure 5.14: Training history of jazz time model for the accompaniment tracks.

The merged model achieved an AUC score of 1, while the last value of the loss function was 0.0159. This indicated that eventually the accompaniment's structure was modelled accurately.

5.2 Popularity prediction

The outcomes of the networks can not be evaluated completely objectively, due to the fact that taste in music is a subjective matter. Thus, an attempt was made to predict the likelihood of the pieces created becoming popular in Spotify platform. For the task of popularity prediction, SpotGenTrack Popularity Dataset [57] was employed. This dataset contains data sources and features extracted. In this thesis, low level features, like chromagram and MFCCs, were used, as they can be extracted from any audio using librosa library [58].

Dataset's songs were categorized into popular and non-popular, based on their score. Therefore, songs with a popularity score above 50 were considered as popular, while the rest non-popular. As a result, this was mainly a classification task. The features were transformed with Linear Discriminant Analysis, using Scikit-learn library [59], while Stochastic Gradient Descent was employed to classify the songs.

The classifier achieved an accuracy score of 73%. Its confusion matrix is displayed in figure 5.15. Zero stands for the non-popular class, while one for the popular. The classification

report is in table 5.1.

	Precision	Recall	F1-score
0	0.80	0.65	0.72
1	0.69	0.83	0.76

Table 5.1: Classification report.

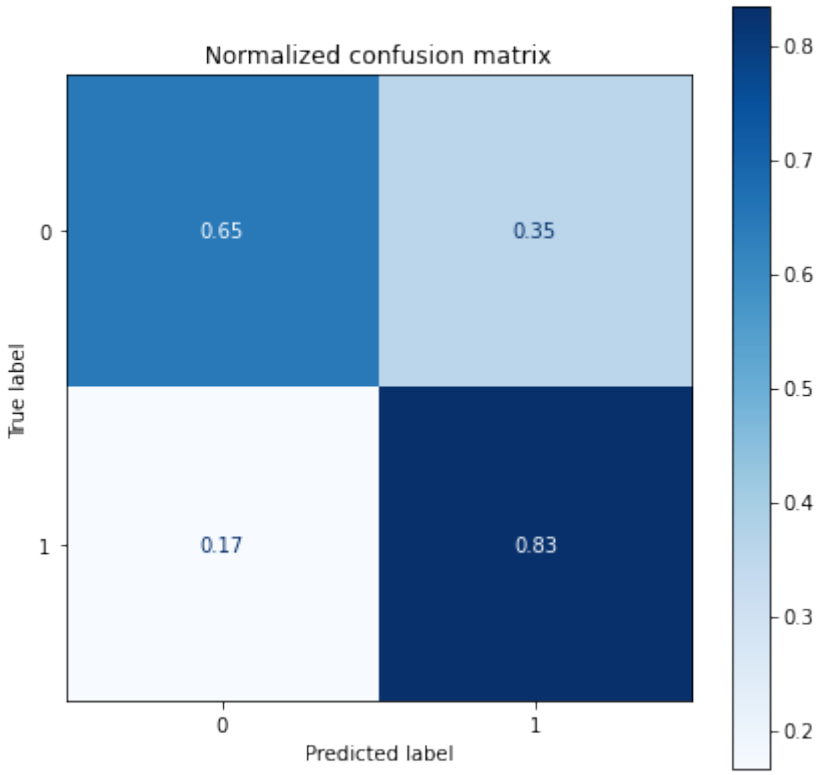

Figure 5.15: Classifier's confusion matrix

5.3 Discussion

The architecture was very successful in generating classical pieces. The outcomes are pleasant to hear and seem to be harmonic. In general, classical music tends to have clearer texture compared to other genres and is mainly harmonic. Therefore, the good performance of the model was somehow expected.

Regarding the blues genre, the model was also successful in generating pleasant and harmonic content. Blues' texture is mostly a single melodic line, consequently it was feasible for the model to learn the melodic and harmonic patterns.

Even though the network reached high AUC scores when trained on the rock dataset, the outcomes are not as good as expected. The pieces composed are pleasant to hear and seem to have harmony, however they would not be identified as rock songs. As a matter of fact, this was expected on the grounds that rock genre has heterogeneity in its pieces. On that account, when a model is trained on a very diverse dataset, the outcomes are expected to be confused.

Jazz genre generally is considered to have very complex harmonic structure. Therefore, the moderate performance of the accompaniment model for Jazz music can be explained. However, both melody and accompaniment models performed well at learning polyphonic jazz content, despite the fact that the pitch model performed poorly with the accompaniment tracks.

As far as the popularity is concerned, the accuracy score of the classifier along with the fact that the genres considered are generally not popular, indicate that it can not be used for objective evaluation. For example, classical pieces are more structured and harmonic compared to the other genres, yet they are classified as non-popular. It is expected, though, as the majority of users do not listen to classical music.

In general, the models managed to generate content that has harmonic structure and is quite expressive. However, they are not comparable to human composed music, regarding long-term structure. Music composed by humans usually has profound structure from the beginning to the end. The networks have captured successfully the short-term structure, meaning that short intervals of the songs are well composed. However, the beginning of a piece generated may have no relation to the notes towards the end.

Since taste in music is a subjective matter, adding that in automated music generation each work considers different metrics of evaluation according to the approach, the results obtained cannot be compared to many works. In [15], the RNN-RBM model achieved the highest accuracy score of 75.40%. In [60], the Bidirectional LSTM model achieved an average accuracy score of 50%. In [23], where another Bidirectional LSTM model is implemented, they achieved an accuracy score of 89% after 15 epochs and 99% accuracy after 30 epochs. In terms of accuracy, the results obtained in the present work are comparable to [23], since the AUC scores are high, indicating also high accuracy. Other works either use different metrics that cannot be compared, like [17] which employs log likelihood, or evaluate their results based on music taste.

Chapter 6

Conclusion

6.1 Summary

The purpose of this thesis was to create an automated music generation system. Towards that purpose, the music content was encoded into two-dimensional arrays containing the velocity values for each tick and pitch. In terms of generating new content by learning musical patterns, the approach of considering sequences in the axes of data was followed.

Two different neural networks were built, using mostly Bidirectional LSTM layers in order to learn based on previous timesteps and capture the long-term structure of music. The first network learns patterns in the vertical dimension, namely the pitch axis, to model the chords, while the second one considers sequences in the horizontal dimension so as to model the temporal sequences along with the notes' dynamics. The two networks were pre-trained and merged into one, resulting in a network that makes predictions for notes, based on the two axes.

This architecture was employed for building two separate models, the melody and the accompaniment model. The melody model, as the name suggests, is responsible for learning and generating new melodies. On the other hand, the accompaniment model is used for generating accompaniment content, in respect to the melody, for various instruments, such as bass and drums.

Melody and accompaniment models were tested for different genres of music, classical, jazz, blues and rock in order to create polyphonic music. Generally the resulting pieces sound harmonic, expressive and have solid short-term structure. Notes occurring simultaneously and small intervals of successive notes seem to have interaction, thus can be considered

harmonic. Regarding long-term structure, it is not captured effectively, due to the fact that a piece may not sound consistent from the beginning to the end.

Moreover, an attempt was made to provide a more objective evaluation metric for music. For that reason a classifier was used that would predict whether a song could become popular in Spotify platform. The classifier was created, however its predicted scores do not actually reflect pieces' quality. As most of the genres considered are not very popular in this platform, the generated pieces can not be judged based solely on the specific classifier's popularity scores.

6.2 Future work

Regarding future work, a fundamental issue is to employ larger datasets of each genre. Neural networks tend to be data-demanding, so the more data you feed them, the better the outcomes. The performance of the proposed methods is expected to get improved when the amount of data used for training would be increased.

The main drawback of the current architecture is the lack of long-term structure. Thus, future work will experiment with the length of the sequences considered by the models towards capturing the structure, hence generating more consistent content.

Last but not least, future work will try to enhance the popularity prediction task. That could be accomplished by experimenting with different datasets containing information from other streaming platforms, where the genres considered in this thesis are more popular. Furthermore, future work will explore considering other features, such as features that indicate whether a song is energetic, dancing e.t.c., instead of low audio features.

Bibliography

[1] Artificial neural network - building blocks. https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_building_blocks.htm.

[2] Jean-Pierre Briot, Gaëtan Hadjeres, and Francois Pachet. Deep learning techniques for music generation - a survey. pages 17 – 40, 09 2017.

[3] Curtis Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, MA, USA, 1996.

[4] David Cope: The Algorithmic Composer. *Computer Music Journal*, 25(2):70–72, 06 2001.

[5] David Cope. *Computer Models of Musical Creativity*. The MIT Press, 2005.

[6] Gerhard Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.

[7] J.D. Fernandez and F. Vico. Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582, Nov 2013.

[8] Alex McLean and Roger T. Dean, editors. *The Oxford Handbook of Algorithmic Music*. Oxford University Press, 2018.

[9] A generative grammar for jazz chord sequences. *Music Perception: An Interdisciplinary Journal*, 2(1):52–77, 1984.

[10] Kemal Ebcioğlu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.

[11] Rebecca Fiebrink and Baptiste Caramiaux. The machine learning algorithm as creative musical tool, 2016.

[12] Peter M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[13] Michael C. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994.

[14] Geoffrey E Hinton, Terrence J Sejnowski, et al. Learning and relearning in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(282-317):2, 1986.

[15] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription, 2012.

[16] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: learning expressive musical performance. *Neural Computing and Applications*, 32(4):955–967, Feb 2020.

[17] Daniel D. Johnson. Generating polyphonic music using tied parallel networks. In João Correia, Vic Ciesielski, and Antonios Liapis, editors, *Computational Intelligence in Music, Sound, Art and Design*, pages 128–143, Cham, 2017. Springer International Publishing.

[18] Hervé Bourlard, Yochai Konig, and Nelson Morgan. Remap: Recursive estimation and maximization of a posteriori probabilities in connectionist speech recognition. In *Fourth European Conference on Speech Communication and Technology*, 1995.

[19] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional lstm networks for improved phoneme classification and recognition. ICANN'05, page 799–804, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] Yuchen Fan, Yao Qian, Feng-Long Xie, and Frank K Soong. Tts synthesis with bidirectional lstm based recurrent neural networks. In *Fifteenth annual conference of the international speech communication association*, 2014.

[21] Hyungui Lim, Seungyeon Rhyu, and Kyogu Lee. Chord generation from symbolic melody using blstm networks, 2017.

[22] T. Jiang, Q. Xiao, and X. Yin. Music generation using bidirectional recurrent network. In *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, pages 564–569, 2019.

[23] Syeda Sarah Azmi, CS Shreekara, and Shwetha Baliga. Music generation using bidirectional recurrent neural nets. *A' A*, 1:S0, 2020.

[24] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[25] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[26] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. 08 2020.

[27] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music, 2020.

[28] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[29] Jason Brownlee. *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Machine Learning Mastery, 2016.

[30] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[31] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In José Mira and Francisco Sandoval, editors, *From Natural to Artificial Neural Computation*, pages 195–201, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[33] B. L. Kalman and S. C. Kwasny. Why tanh: choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 578–581 vol.4, 1992.

[34] Jonathan Barzilai. Chapter 15 - on neural-network training algorithms. In William F. Lawless, Ranjeev Mittu, and Donald A. Sofge, editors, *Human-Machine Shared Contexts*, pages 307 – 313. Academic Press, 2020.

[35] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014.

[36] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[37] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[38] George Hinton. Overview of mini-batch gradient descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[39] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[40] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., USA, 1st edition, 1999.

[41] Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157–66, 02 1994.

[42] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[43] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, Feb 2005.

[44] Sarang Narkhede. Understanding auc-roc curve. *Towards Data Science*, 26:220–227, 2018.

[45] Official midi specifications. `https://www.midi.org/specifications`.

[46] Musical instrument digital interface (midi). `https://www.recordingblogs.com/wiki/musical-instrument-digital-interface-midi`.

[47] Rui Guo, Dorien Herremans, and Thor Magnusson. Midi miner – a python library for tonal tension and track classification. 10 2019.

[48] Giles Hall. Python-midi. `https://github.com/vishnubob/python-midi/`.

[49] G. Read. *Music Notation: A Manual of Modern Practice*. A Crescendo book. Allyn and Bacon, 1969.

[50] Claude Sammut and Geoffrey I. Webb, editors. *Binning*, pages 140–140. Springer US, Boston, MA, 2017.

[51] Dynamics. `https://musescore.org/en/handbook/3/dynamics`.

[52] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[53] François Chollet et al. Keras. `https://keras.io`, 2015.

[54] Netron: Visualizer for neural network, deep learning and machine learning models.". `https://www.lutzroeder.com/ai`.

[55] Soumik Rakshit. Classical music midi, May 2019.

[56] Vee Upatising. Blues genre midi melodies, Feb 2020.

[57] David Martín-Gutiérrez. SpotGenTrack Popularity Dataset, June 2019. type: dataset.

[58] Brian McFee, Vincent Lostanlen, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Jack Mason, Dan Ellis, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, Keunwoo Choi, viktorandreevichmorozov, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao

Wei, nullmightybofo, Darío Hereñú, Fabian-Robert Stöter, Pius Friesch, Adam Weiss, Matt Vollrath, and Taewoon Kim. librosa/librosa: 0.8.0. https://doi.org/10.5281/zenodo.3955228, July 2020.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[60] Hyungui Lim, Seungyeon Rhyu, and Kyogu Lee. Chord generation from symbolic melody using blstm networks, 2017.