POLITECNICO DI TORINO

MASTER OF SCIENCE IN ICT FOR SMART SOCIETIES

MASTER'S DEGREE THESIS

Machine Learning Based Prediction of MIDI Signals for Networked Music Performance Applications

Supervisors:
Dott.ssa Cristina ROTTONDI
Prof. Andrea BIANCO

Candidate:
Paolo GRASSO

DECEMBER, 2020

ABSTRACT

Networked Music Performance (NMP) is envisioned as a potential game changer among Internet applications: it aims at revolutionising the traditional concept of musical interaction by enabling remote musicians to interact and perform together through a telecommunication network.

Ensuring realistic performative conditions, however, constitutes a significant engineering challenge due to the extremely strict requirements in terms of audio quality and, most importantly, network delay. Unfortunately, such requirements are rarely met in today's Internet. When an audio signal is streamed from a source to a destination, audio data is divided into packets; the delivery of packets to the destination is subject to an unavoidable delay due to transmission and propagation over the physical medium, plus a variable jitter. Therefore, some or even most packets may not reach the destination in time for the playback, which causes gaps in the audio stream to be reproduced.

This thesis proposes the adoption of machine learning techniques to conceal missing packets carrying MIDI audio signals, by predicting future events that will be generated by the MIDI source.

Results show that the proposed approaches based on feedforward and recurrent artificial neural networks increase performance up to 20% in term of proposed metrics with respect to a baseline model whose task is repeating the last played notes given as input.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all the people who have helped me with my thesis; in particular, I would like to thank Alessandro Giusti and Juan Andrés Fraire for assisting me throughout the project and giving me advice and suggestions for my work.

I want to thank all my friends who have been with me during these years; a big thank goes to Marco and Matteo, with whom I have shared wonderful experiences and who have always believed in me from the very beginning of this journey.

A very special thank goes to my dear Francesca with whom I have spent a lot of beautiful moments and shared most of the projects over these years; thank you for being with me throughout this adventure.

A final thank goes to my family and my relatives who have always been proud of me and have always encouraged me throughout my studies; all your support has been very appreciated. Thank you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 MOTIVATION

Networked Music Performance (NMP) is a type of music performance where musicians play together from different places and music is transmitted over a telecommunication network to simulate that they were in the same room. This can be useful for different activities: let's think about a music school where educators could teach students from distance, or a music band whose members could rehearse together from their homes, or even live stream a show.

Such approach requires very low latency transmissions with the minimum amount of errors since the music track has to be played and reproduced in real-time for all musicians. They, in fact, should be able to keep up with the music playback in order to be synchronised with each other and execute their parts in the exact same moment. It is very crucial for musicians to listen to music parts generated by the other members they are playing with, since they have to fit together the arrangement of the song, and it would not be doable without a constant feedback from other players.

The main problem is that telecommunication networks like Internet are known to be unreliable; it is not possible to build up a reliable infrastructure based only on such networks since they are very subjected to congestions, jitters and delays which can easily make the networked performance infeasible. Standard protocols and algorithms for error correction and packet retransmission are not well suited since they cannot keep up with the real-time playback of a song, due to the introduction of delay to perform the error correction procedures. Some protocols, like User Diagram Protocol (UDP), succeed in maintaining the real-time flow by ignoring errors and disregarding packet retransmissions; however, when packet losses and delays occur, the musicians would not be able to listen to the music stream and they would be forced to play without a feedback until the stream is resumed.

A possible solution to address these problems can be researched in developing a framework in which the music stream is coded and decoded into Musical Instrument Digital Interface (MIDI) format, which drastically reduces the amount of data to be transmitted with respect to a stream of real sounds, and therefore reduces the possibility of congestions and delays. With MIDI protocol, music data can be coded into very lightweight structured archive files, which include information like notations, pitches, velocity, vibrato, stereo panning and clock signals of the track. The small size of these archives lies in the fact that they do not contain audio recordings, but only a transcription of the track, similar to a digital sheet music.

Such extremely small sizes of MIDI signals compared with raw audio signals make them suitable for applications like NMP where transmitting a low amount of data can be crucial to preserve bandwidth and avoid congestions.

In spite of this, it is not realistic to assume a completely fluid transmission of signals without incurring in packet losses which cause gaps in the playback.

1.2 THESIS OBJECTIVES

This thesis focuses on the development of a machine learning model based on artificial neural networks to fill audio playback gaps generated by transmission errors, in order to ensure a continuous music stream without interruptions. Such model is able to take as input the MIDI events occurred in a window of several timesteps and return as output the predictions about MIDI events that are likely to occur in future timesteps. Hence, possible gaps in the playback are filled using predictions which the model generates based on the audio data received in the time instants previous to the moments when errors occur. This can be exploited for different applications: in particular, the model could be integrated with a streaming system to improve the quality of the playback. Therefore, the proposed model does not aim to be a stand-alone system already functional to reproduce the audio data streamed during a networked music performance, but a support to improve an already existing audio streaming system.

Performance is evaluated using commonly adopted metrics such as F1 score, AUC-ROC and a custom metric based on values which quantify the consonance perception to human hearing. In all cases the proposed model outperforms a simple baseline model whose task is repeating the last played notes from the timestep in which error occurs.

1.3 THESIS OUTLINE

The rest of the chapters are organised as follows: chapter 2 describes the state-of-the-art; chapter 3 outlines basic concepts used in developing the model such as data representation and artificial neural networks; chapter 4 describes in more detail the choices made to implement the model and the different parameters which have been studied and selected; chapter 5 discusses the numerical assessment of the proposed prediction models; in the end chapter 6 contains a conclusive summary.

2 RELATED WORK

This chapter describes the history of networked music performance during the last decades and discusses some works and publications which explore machine learning techniques in order to analyse and describe features of musical data.

2.1 HISTORY OF NETWORKED MUSIC PERFORMANCE

The concept of a networked music performance has spread and has been experimented over the last century with the diffusion of electronic devices, especially personal computers. The first real experiment was performed in 1951 by John Cage using radio transistors as musical instruments by interconnecting them and so influencing each other [1].

In the late 1970s, *The League of Automatic Music Composers* was one of the first music ensembles to investigate the unique potentials of computer networks as a medium for musical composition and performance, by creating novel forms of music with analog circuits and interconnected computers.

In the 1990s, a more sophisticated experiment by *The Hub* band consisted in transmitting MIDI signals over Ethernet to distributed locations via a MIDI-hub which was functioning as a switchboard for routing those signals to connected musicians [2]. The choice of transmitting MIDI data instead of audio signals was due to technology and bandwidth limitations of that period, but when Internet became available to everyone with the World Wide Web, people started to think about new possible ways to perform music concerts with musicians connected from different locations.

From the 2000s, more developments of high-speed connections over Internet made high quality audio streaming possible. Researchers made new experiments by taking advantage of improved network conditions: at Stanford University's CCRMA, *SoundWIRE* (Sound Waves on the Internet for Realtime Echoes) created a project that consisted of transmitting ping signals as sonar-like sounds in order to 'hear' the qualities of bidirectional Internet connections [3]. In the wake of SoundWIRE experiments, several research groups developed their own systems for networked music performances: Distributed Immersive Performance (DIP) studies the technologies to recreate high fidelity channels for audio and video to generate virtual spaces in which immersive performances take place [4]; SoundJack is a software which allows low latency audio/video communications on standard computer systems without particular hardware requirements [5]; DIAMOUSES is an open framework whose objective is to enable a wide range of scenarios including master teaching, show rehearsing, jamming sessions and collaborations, for networked music performance [6, 7].

2.2 WORKS RELATED TO THIS THESIS

This section explores some article and publications which are somehow related to the purpose of this thesis.

The article [8] proposes an artificial neural network model based on Long Short-Term Memory (LSTM) networks for prediction of polyphonic MIDI sequences. The article is focused on developing a model which can be used for automatic music transcription which is very similar to speech recognition where *language models* are combined with *acoustic models* to transcribe spoken words into written text. By applying the same concept it is possible to transcribe music into different representations like MIDI files or *piano rolls*. The study evaluates the impact of various parameters on the predictive performance of the system; these parameters include the number of hidden nodes, the learning rate, the sampling rate of the piano roll and data augmentation. Performance is evaluated using different metrics as precision, recall and F1 score. Results show that the speed of the training phase of the model is influenced by those parameters, especially the learning rate and number of hidden nodes, while the quality of prediction is influenced by the sampling rate, since in the piano rolls there are more self-transitions, i.e. prolonged notes from previous timesteps.

Reference [9] proposes a neural network model whose focus is learning interval representations of musical sequences in *monophonic* music. This kind of model has the ability to perform copy-and-shift operations in order to repeat musical pattern and transpose melodies. This comes in hand to the problem of decoupling absolute pitches and music regularities which is usually overcome by performing data augmentation on the training dataset. The model uses a *recurrent gated auto-encoder* (RGAE), which is a recurrent neural network able to operate on *interval representations* of musical sequences. By learning all intervals within a window of n pitches, the system is more robust to *diatonic* transpositions. Results about a pattern repetition show that the RGAE provides a significant improvement over a standard Recurrent Neural Network (RNN) approach, with a prediction accuracy of above 99% over the 42% provided by RNN.

In the article [10] the *Pulse* framework is used to discover a set of revelant music features which can be utilised for sequential prediction of symbolic *monophonic* music. Two different types of features are discovered: *viewpoint* features which indicate the presence of a specific sequence of events, and *anchor* features which represent the concepts of tonic and mode (key) in common tonal music. Viewpoint features are generalised versions of classical n-grams: they do not necessarily contain contiguous sequences of basis features, and they may be composed of basis features derived from different alphabets. Anchor features are defined with respect to an anchor tone and not with respect to the previous tone, and therefore they carry information from the very first tone in the piece. The training of the model is also helped by a *regularisation* process which prevents overfitting by limiting the growth of the feature set. The system is then combined with different approaches including *long-term memory* where the model learns the characteristics of an entire dataset, *short-term memory* where the model learns the properties for a single piece of the dataset, and a hybrid version of the two methods. Results

show that all approaches outperform the current state-of-the-art models on a standard benchmark corpus of folk songs and chorales by Johann Sebastian Bach.

In reference [11] the concepts of consonant and dissonant perception generated by two-tone intervals between music notes are studied from a mathematical point of view. It is known that in Western culture, the accepted order of 'perfection' is the one proposed by Hermann Helmholtz in [12], which is reported in table 2.1. The value of ΔQ represents the width of the stability of a particular interval between two tones within an octave. The widest stability is reached with the *unison*, where two tones have the same pitch, while the smallest stability is reached with *minor second* and *tritone* which are so dissonant that their ΔQ is extremely difficult to evaluate. Lots and Stones point out several problems with Helmholtz's theory, arguing that 'it remains controversial and fails to explain a number of non-trivial aspects central to musical psychoacoustics'. However, even though the purpose of their work was to criticise the use of Helmholtz resonance theory in cases where it cannot be applied coherently, the reported table gives a light description of what intervals are considered more consonant, for example fifths and fourths are more pleasant to hear with respect to major sevenths and minor sevenths, and therefore this can be exploited to create a basic evaluation metric for music prediction models.

Interval's evaluation	Interval's name	Interval's ratio	ΔQ
Absolute consonances	Unison	1 : 1	0.075
	Octave	1 : 2	0.023
Perfect consonances	Fifth	2 : 3	0.022
	Fourth	3 : 4	0.012
Medial consonances	Major sixth	3 : 5	0.010
	Major third	4 : 5	0.010
Imperfect consonances	Minor third	5 : 6	0.010
	Minor sixth	5 : 8	0.007
Dissonances	Major second	8 : 9	0.006
	Major seventh	8 : 15	0.005
	Minor seventh	9 : 16	0.003
	Minor second	15 : 16	—
	Tritone	32 : 45	—

Table 2.1: Width of stability for two-tone intervals according to Helmholtz

3 BACKGROUND

This chapter describes general concept about music data representation and artificial neural networks which will be covered in the following chapters.

3.1 MUSICAL DATA

3.1.1 Piano roll representation

For this thesis, musical data consists in music pieces which are stored in MIDI files whose standard is defined in [13]. Each MIDI file contains notes whose pitches are defined by a number from 0 to 127 according to table 3.1, along with other parameters like duration and velocity. All those numbers can be converted to a *piano roll* representation: a piano roll is a $P \times T$ matrix M where $M(p, t)$ denotes if a note of pitch p is played at timestep t. As said before, P is equal to 128, equivalent to the number of pitches in a MIDI file, but the piano roll can be cropped by reducing P to a value of 88, in order to include only the grand piano keyboard $[A_0, C_8]$, which correspond to MIDI numbers $[21, 108]$ or even smaller size keyboards, for example 64 or 49 keys.

Every cell of the piano roll has a value in range $[0, 127]$ which represents the *velocity* of the note. Velocity corresponds to the intensity of the played note, i.e. how loud it sounds. By replacing all non-zero velocity values with 1s it is possible to obtain a binary piano roll which can be more suitable as input to a machine learning algorithm, since volume dynamics are removed from the list of features.

Octave	C	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B
−1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Table 3.1: MIDI note numbers. A standard piano keyboard covers the range of notes $[A_0, C_8]$.

3.1.2 Beat resolution

Notes in musical pieces can assume different values or durations which contribute to the rhythm of the song; the most common are reported in table 3.2.

Figure 3.1a shows a three-measure sheet music that has been coded into a MIDI file and then converted into a piano roll with different beat resolutions. It is visible that a beat resolution of 10 is not enough to represent the *demisemiquavers* which are played beats 1–2. Also the *semiquavers* in beats 3–4 seem to be represented with some inaccuracy.

A beat resolution of 16 or 24 is enough to preserve all these types of notes; a resolution of 16, however, introduces a lot of silence among the notes which does not appear when using a higher resolution such as 24. The resolution must be set in order to prevent smaller notes to disappear and also to limit an excessive number of timesteps which implies single notes to appear in several consecutive occurrences.

Name	Relative value	Symbol
semibreve	1	𝅝
minim	1/2	𝅗𝅥
crotchet	1/4	♩
quaver	1/8	♪
semiquaver	1/16	𝅘𝅥𝅯
demisemiquaver	1/32	𝅘𝅥𝅰

Table 3.2: Common note duration values

3.1.3 Datasets

The datasets used for predictions are collections of musical pieces stored in MIDI files [14], already divided into training, validation and test sets. Different collections are available, including:

CLASSICAL PIANO Archive of classical piano songs; Total duration: 8:20:29

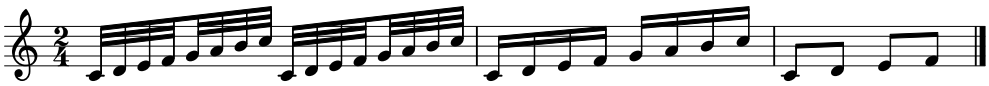
NOTTINGHAM Collection of 1200 folk tunes; Total duration: 18:34:42

JSB CHORALES Entire corpus of 382 four-part harmonized chorales by Johann Sebastian Bach; Total duration: 3:51:20

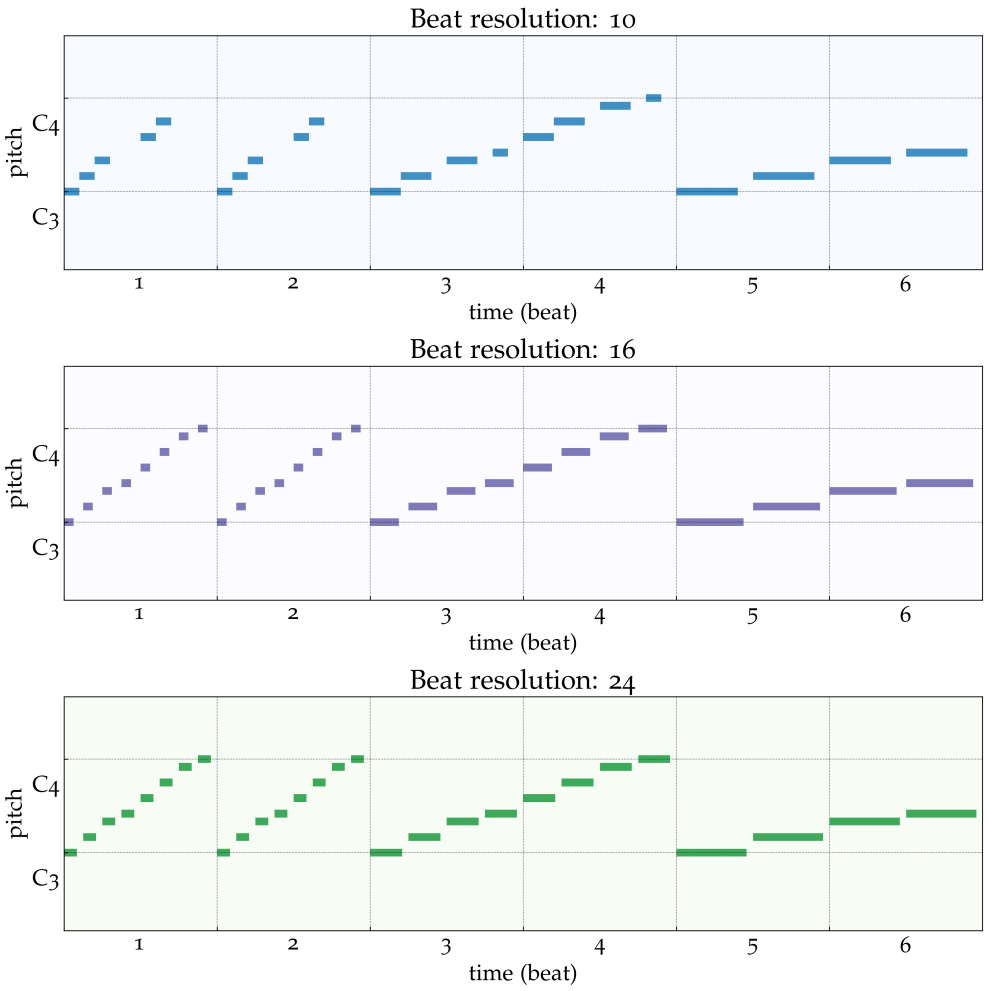
Nottingham and J. S. B. Chorales contain pieces which are similar to each other from a melodic point of view, whereas such homogeneity is less detectable in the Classical Piano dataset.

3.2 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

The concept of Artificial Intelligence (AI) describes the ability of a machine to imitate human beings, perceive environmental events, learn from errors and maximise the possibility of achieving a certaing goal. Fields that are



(a) Sheet music from which the MIDI file was generated



(b) Piano rolls

Figure 3.1: Comparison among the same MIDI file converted into a piano roll representation with different beat resolutions

included in AI are generally relative to tasks that are considered out of ordinary, for example understanding human speech or creating self-driving vehicles. Other tasks, like optical character recognition, have become available for everyone in routine technologies and they are actually excluded from applications considered to be AI [15].

One subset of AI is Machine Learning (ML), by which algorithms are trained on specific datasets in order to create models which can learn from available data and know how to behave when encountering previously unseen data. In ML there are different approaches in teaching a model to behave in a certain way: in the *supervised learning*, the model learns how to distinguish data by knowing in advance many examples in which solutions are provided; in the *unsupervised learning*, the model learns to distinguish among different classes by only analysing the features of data, without knowing in advance any examples with provided solutions. ML includes a large variety of applications like pattern recognition, image elaboration, computer vision and genetic algorithms [16].

A particular subset of ML is the Deep Learning (DL) where learning methods are based on *artificial neural networks*; the adjective 'deep', in fact, refers to the multitude of layers that those networks are made of.

The relationship among AI, ML and DL is depicted in fig. 3.2. This representation comes from the concepts explained above, which are very common definitions but they are not universally accepted: other lines of thoughts in fact tend to consider AI and ML as two big sets which intersect and DL occupying part of the intersection and part of ML. To summarise:

ARTIFICIAL INTELLIGENCE Any technique that enables computers to learn and reason like humans.

MACHINE LEARNING Subset of AI techniques that use algorithms and statistical methods to enable machines to learn with experience.

DEEP LEARNING Subset of ML in which multi-layer neural networks adapt themselves and learn from vast amounts of data.

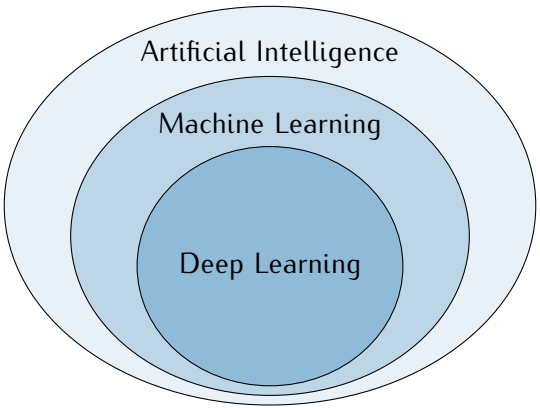


Figure 3.2: Commonly accepted relationship among artificial intelligence, machine learning and deep learning

3.3 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are computing systems inspired by biological neural networks of animal brains; they try to vaguely simulate the architecture of a brain, so they are constituted of a series of *layers* with multiple nodes called *artificial neurons*. The nodes of each layer are connected to the nodes of other layers by connections called *edges*. Each neuron is usually associated with a weight which is adjusted with learning; a weight alters the strength of the signal at an edge. Moreover, neurons can have an *activation function* which modifies the emitted signal, for example only if it is above a certain threshold.

Each layer of the network represents a transformation of the input which is then sent to the next layers; the overall transformation from the input layer to the output layer constitutes the mathematical function of the neural network.

3.3.1 Artificial neurons

Artificial neurons are the elementary units of neural networks. They consist in a mathematical function which aggregates and processes the signals received as input and returns an output which is sent to other neurons. A graphical representation is depicted in fig. 3.3.

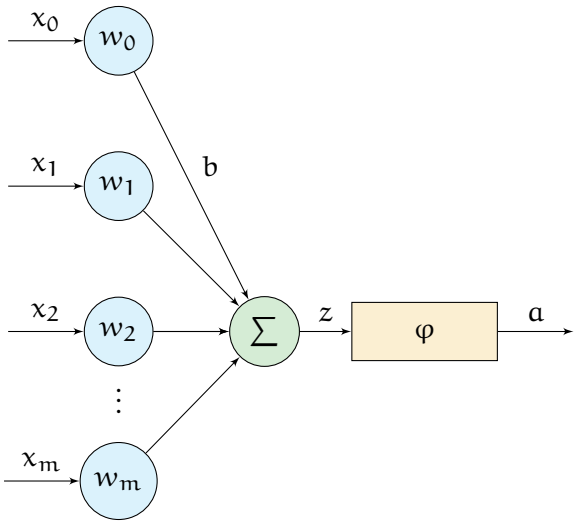


Figure 3.3: Functioning of an artificial neuron

For a given neuron, let there be a set of m inputs with signals x_1 through x_m and the respective sets of weights w_1 through w_m. The output a of the neuron can be described by eq. (1), where φ is the activation function which modifies the strength of the output signal. The variable $b = w_0 x_0$ is a *bias* associated to the neuron which shifts the decision boundary away from the origin and does not depend on input values.

$$a = \varphi\left(\sum_{j=0}^{m} w_j x_j + b\right) \tag{1}$$

3.3.2 Activation functions

Activation functions are another key element of artificial neural networks since they are necessary to make the network model *nonlinear*, thus making it possible to solve non trivial problems with a small amount of neurons. Common activation functions are shown in fig. 3.4. These functions are used in hidden layers and output layers for different purposes, e.g. output layers usually have an activation function which fits the machine learning task: multi-class classification will benefit from a softmax activation since only one class can be correct at a time, whereas multi-label classification will benefit from the sigmoid which allows several classes to be selected at the same time.

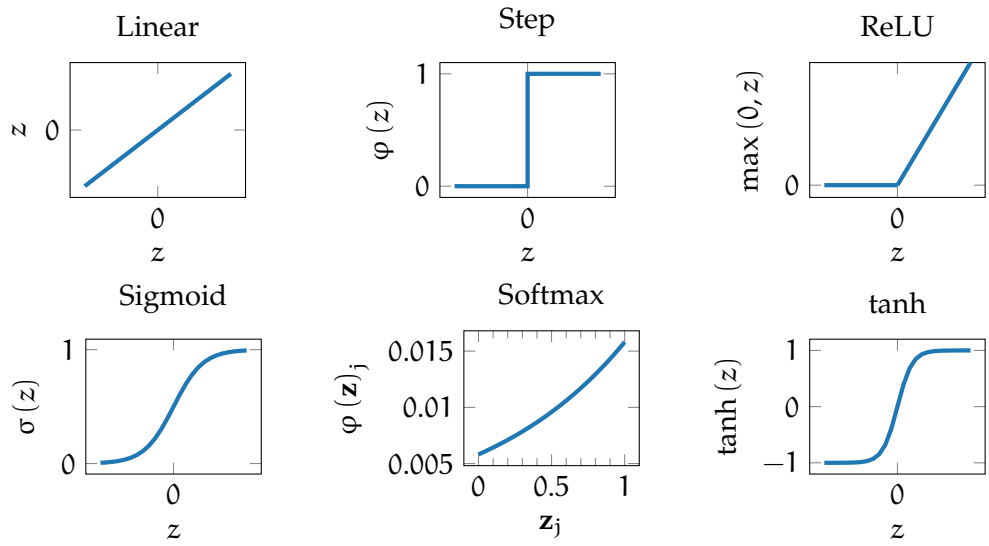


Figure 3.4: Common activation functions in neural networks. Softmax example is relative to a vector $\mathbf{z}$ made of 100 equidistant elements in range $[0, 1]$.

3.3.3 Loss function, backpropagation and optimisers

The weights of the neurons are calibrated by a *loss* (or *cost*) function which compares the network output values with the *real* output values (labels) and returns a value which tells 'how bad' the transformation from input to output was made. This loss value is used to adjust the weights of the neurons; the process is executed by an algorithm called *backpropagation* which basically computes the *gradient* of the loss function and provides weight adjustments from the end to the beginning of the network.

A really basic loss function is the quadratic cost function, shown in eq. (2). It evaluates the quadratic difference between the network's output a and the desired real output y: the closer the two values are, the lower the loss function is. This kind of loss function, however, has issues regarding the learning speed of the network, since it tends to provide a very slow learning when the two output values are very different, i.e. when the neuron's output is unambiguously wrong. This happens mainly when the activation function is a sigmoid σ, since when computing the gradient, the partial derivative

with respect to the weight depends on the derivative of the sigmoid σ' as shown in eq. (3), which assumes very low values at tails (fig. 3.5), i.e. when a and y values are very different.

$$C = \frac{(y-a)^2}{2} \tag{2}$$

$$\frac{\partial C}{\partial w} = a\sigma'(z) \tag{3}$$

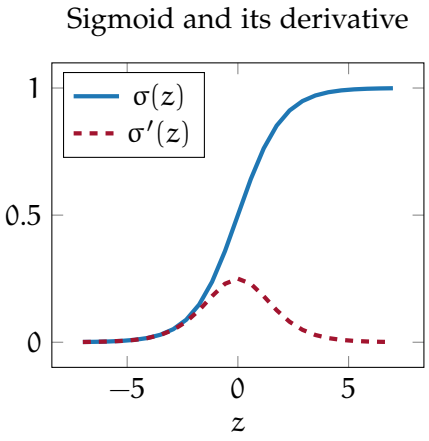


Figure 3.5: Derivative of the sigmoid function

A more advanced loss function which addresses the slowdown issue is the *cross-entropy*, shown for a single neuron with multiple inputs in eq. (4). It assumes values near 0 when the network's output a is very close to the desired output y, and values near 1 when output and desired values are distant. This functions make sure that the learning speed is higher when neuron's output is unambiguously wrong, and lower when it is near the desired output. This is visible by computing the partial derivative of the loss function with respect to the weight, as shown in eq. (5): the learning rate, in fact, depends on $\sigma(z) - y$ which is just the error in the output. [17]

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)] \tag{4}$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y) \tag{5}$$

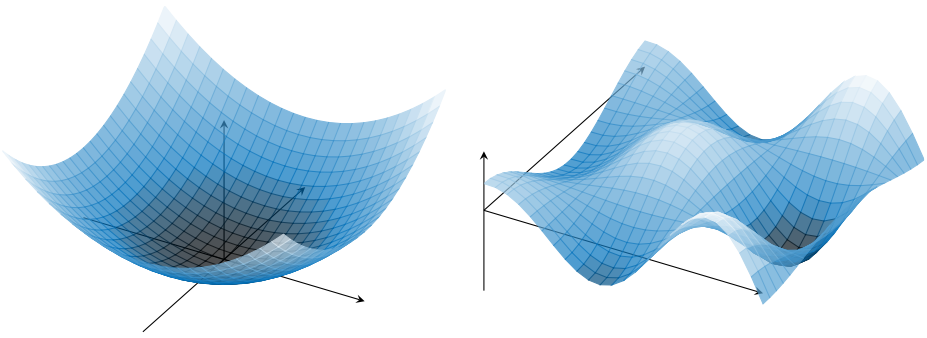
The weight adjustment during backpropagation is based on the gradient descent of the loss function; however, since neural networks are usually very complex, such loss functions are hardly convex with only one minimum as the one depicted in fig. 3.6a, but they have a complex shape with many local minima and a single global minimum, as shown in fig. 3.6b. In order to reach the global minimum, different variations of gradient descent exist: the most commonly used in neural networks are the Gradient Descent with Momentum, AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Propagation), and Adam (Adaptive Moment Estimation) which are called *optimisers* since they optimise the learning rate of the gradient descent and provide a faster learning.

GRADIENT DESCENT WITH MOMENTUM The descent with momentum makes it possible for the gradient to overcome small slopes near local minima by increasing the possibility to reach the global one. It can be seen as a ball that rolls down a slope with a momentum which lets it go uphill for small distances across the track.

ADAGRAD Variation of the stochastic gradient descent. It uses different learning rates for different parameters, by increasing it for sparser ones and decreasing it for ones that are less sparse. It can cause a problem where the network stops learning due to very small learning rate for such parameters.

RMSPROP It optimises the update of parameters by favouring the movement in the direction of the weights and reducing oscillations caused by other parameters.

ADAM It combines the RMSProp method and the descent with momentum for a better optimisation algorithm.



(a) Function with one minimum **(b)** Function with many minima

Figure 3.6: Minima of a 3-D function

3.3.4 Feedforward neural networks

Feedforward neural networks are a type of artificial neural networks where the connections among neurons do not form cycles; therefore information from one layer can only go in one direction to next layers and does not go back. They are considered the first devised and simplest type of artificial neural networks. They do not keep memory of past events: the output of the network depends only on the current input. An example of a feedforward model is depicted in fig. 3.7: it is visible that every layer gets information from the previous layer and sends information to the next one. Layers in the middle are called *hidden* since they are placed between the input and the output; therefore they are not visible to an external system.

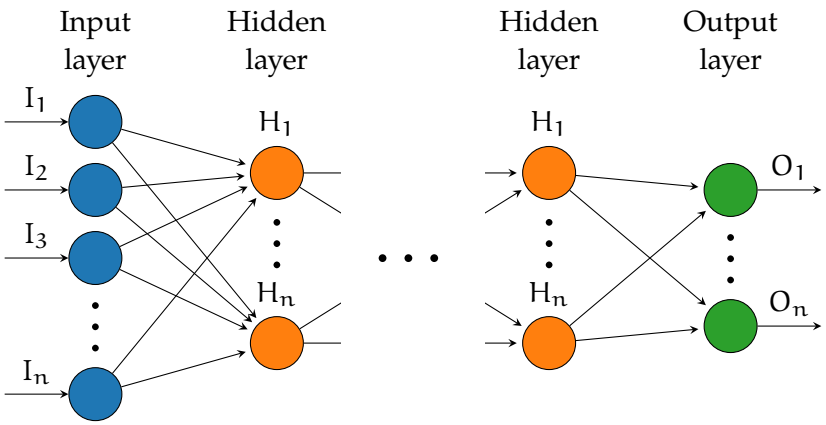


Figure 3.7: Feedforward neural network

3.3.5 Recurrent neural networks

RNNs are a type of artificial neural networks where connections among nodes form a directed graph along a temporal sequence. They have an internal state memory that can be used to process input sequences of variable length. Hence, RNN are particularly used when the dataset is made of data that is inherently sequential, e.g. speech, text, weather, financial, or, in our case, a sequence of notes and chords which compose a song.

An example of RNN is represented in fig. 3.8: in this case the hidden layers keep in memory past information and reuse such information in subsequent calculations. The network therefore has a 'memory' state which is stored inside recurrent *cells* of neurons that increases the learning potential. Several types of recurrent cells exist, which are represented in figs. 3.9 to 3.11 and described below:

SIMPLE RNN It is the simplest recurrent layer; it combines the input with an *hidden state* h which stores information about past data seen by the cell. This state is maintained and used along with the next inputs to keep a sort of connection among all different instances.

GRU (GATED RECURRENT UNIT) It is an upgraded version of SimpleRNN; it features an *update gate* and a *reset gate* which respectively decide when to update or reset the *hidden state* of the system, i.e. which information is maintained from one process to the another.

LSTM (LONG SHORT-TERM MEMORY) It is more effective than SimpleRNN and has some advantages with respect to GRU. Besides the hidden state, it features also a *cell state* c which is relative to the activation functions inside the LSTM cell itself. Moreover, there are a *forget gate*, an *input gate* and an *output gate* which modify the value of hidden and cell states. These three gates are able together to retain even more information when compared with GRU, but the training process is slower due to the more complex system.

In general, the more complex is the recurrent cell, the higher number of parameters are trainable inside it.

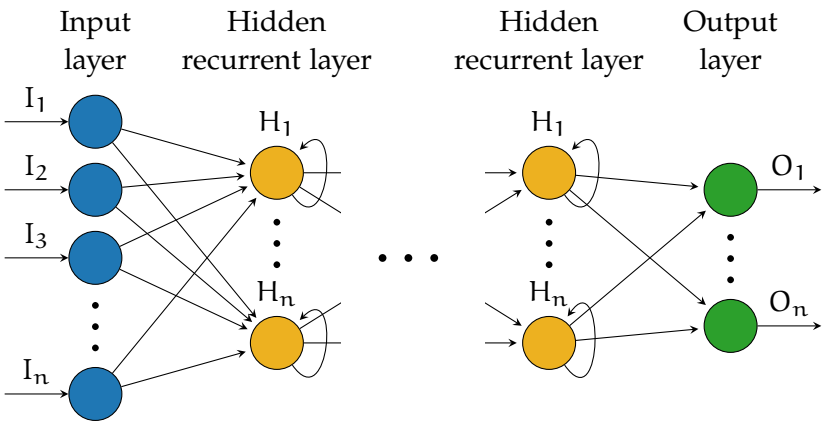


Figure 3.8: Recurrent neural network

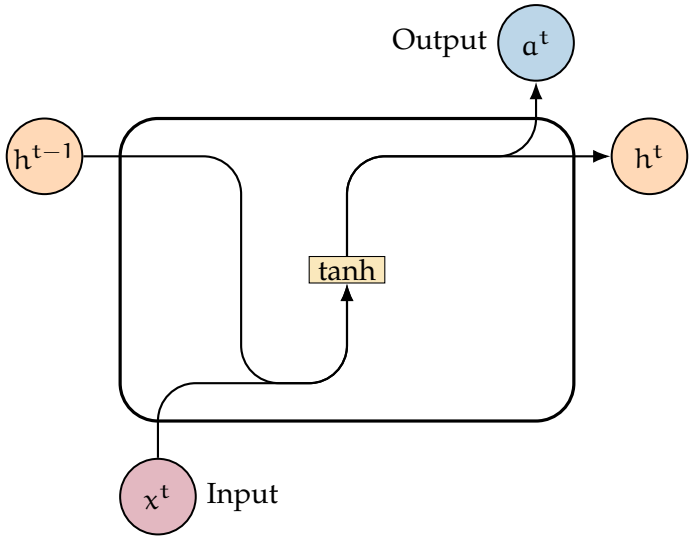


Figure 3.9: SimpleRNN cell diagram. The input x^t and the hidden state h^{t-1} are simply multiplied together and passed through an activation function.

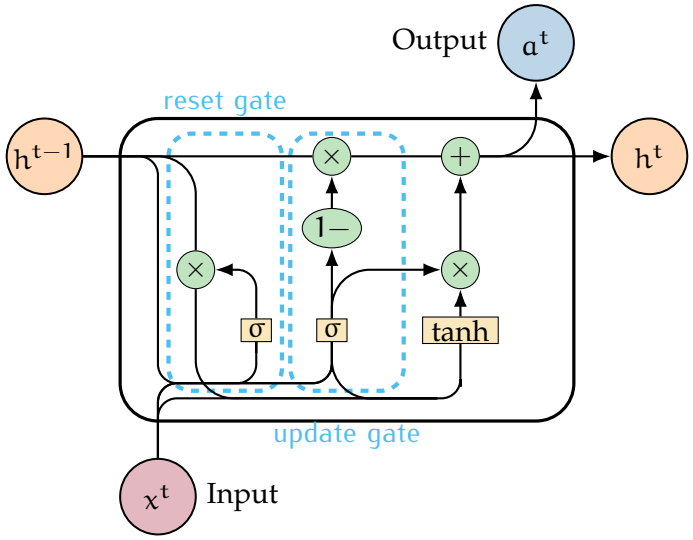


Figure 3.10: GRU cell diagram. An *update gate* decides which information of h^t is passed to the next cell, while a *reset gate* decides how much past information to forget.

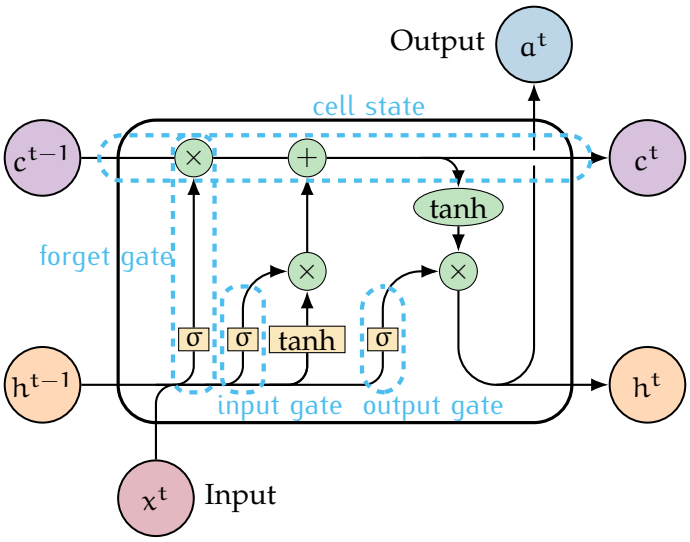


Figure 3.11: LSTM cell diagram. A *cell state* c^t contains information of the dependencies among elements in the input sequence; a *forget gate* decides what information should be kept or forgotten; an *input gate* decides which information is passed to the cell; and an *output gate* decides what the next hidden state h^t should be.

3.3.6 Vanishing and exploding gradients

A common problem present in artificial neural networks is related to the gradient descent used in the backpropagation algorithm. As described in the previous sections, neural networks are generally made of many layers interconnected with different structures. Layers that are deeper into the network go through continuous matrix multiplications in order to compute their derivatives since their neuron's weights are updated by taking into consideration how the weights of connected neurons have been previously updated. Therefore in a network made of n layers, n derivatives are multiplied together: if they are small, the gradient computed at the deeper layers will extremely decrease, causing a *vanishing gradient*; if they are large, the gradient at the deeper layers hugely increases, causing an *exploding gradient*.

VANISHING GRADIENT Neurons become not able to meaningly learn due to extremely low adjustments in their weights which eventually stop the network from training.

EXPLODING GRADIENT Neurons become not able to effectively learn, causing system instability and possible overflow in weights which can no longer be updated due to the appearance of NaN values.

Possible solutions to address the vanishing gradient problem can be: lowering the number of layers, and use the ReLU activation function in hidden layers [18]. Regarding the exploding gradient, possible solutions consist in lowering the amount of layers, and limiting the size of the gradient by rescaling it to a lower value when it becomes too large; this last process is called gradient clipping [19].

4 PROPOSED FRAMEWORK

This chapter describes the framework used to process musical data along with the implementation of artificial neural network models. Recall that in case of missing packets in an audio streaming, error correction methods are not applicable since they would introduce some delay which is not compatible with the purpose of ultra low latency music streaming.

4.1 PROPOSED SOLUTION

4.1.1 System architecture

The solution proposed in this thesis consists in a machine learning model that is able to fill music gaps in the playbacks which occur in correspondence of packet losses or generic transmission errors. The music stream is therefore patched with notes generated by the model and resumed so that the listener does not experience interruptions during the playback. These fillings have to be musically coherent to what the musicians are expecting to hear, possibly by maintaining the correct cadence and rhythm of the song.

The model uses a window of timesteps preceding the timestep at which the error occurs to generate predictions which can be used to patch the streaming, as shown in fig. 4.1.

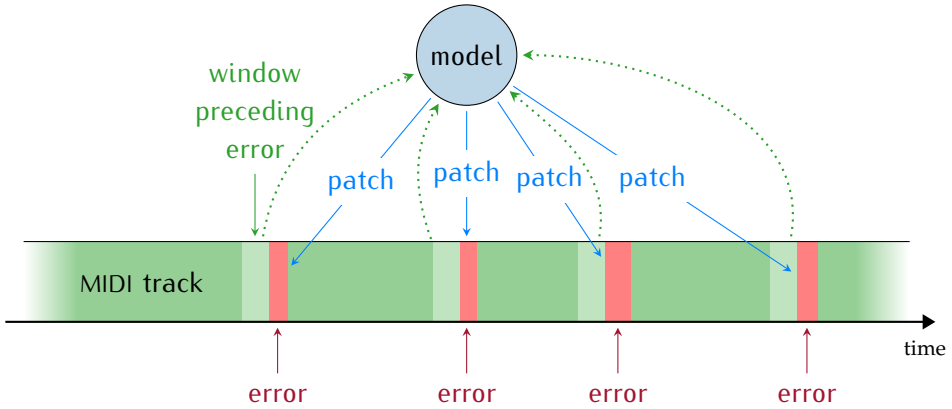


Figure 4.1: Filling gaps in MIDI track

Since the music stream is made of sequences of MIDI events, the original track has to be preprocessed in order to be fed into the neural network model. The track goes through a conversion process that turns it into a piano roll representation. Such piano roll is then given as input to the model, which returns another piano roll containing predictions for future timesteps afterwards the input piano roll. Those predictions can be converted again into a MIDI format which can be played in place of the original MIDI track in

case of packet losses and delays. The flow diagram of the process is depicted in fig. 4.2.

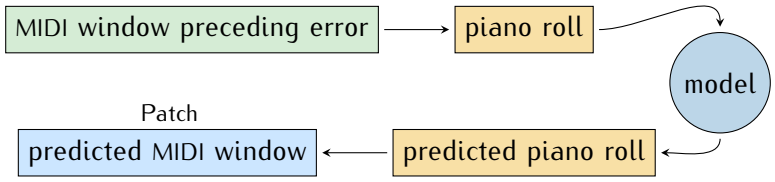


Figure 4.2: Model processing diagram

4.1.2 System parameters

Window size

The model handles windows of timesteps whose size depends on the number of units of the last layer of the neural network, meaning that once the model is trained, it always takes as input and returns as output the same number of timesteps. Moreover, the tempo of the song does not influence the number of notes that are predicted, since it only affects the speed at which the timesteps are reproduced, but not the amount of them. In the proposed experiments, windows sizes of 1 and 10 timesteps are tested.

The process of patching the music stream using windows of 10 timesteps is shown in fig. 4.3: assuming that an error occurs at timestep t_n creating a gap in the playback from that point forward, the model takes the window composed by timesteps from t_{n-10} to t_{n-1} and generates predictions covering a window from t_n to t_{n+9} which can be used to fill the gap in the playback. In this way, the 10 predicted timesteps work as a coverage for the worst case in which the gaps prolongs for several timesteps up to 10. For the case in which the gap is shorter, only the first predicted timesteps are used.

Hence, supposing that during a music stream the model continuously generates windows of predicted timesteps, it is due to the application which uses such model to decide how many timesteps should be taken from each window in order to fill the possible gaps in the playback.

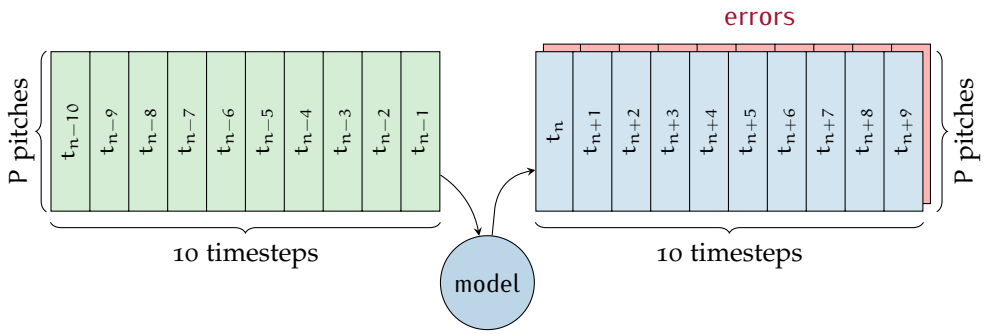
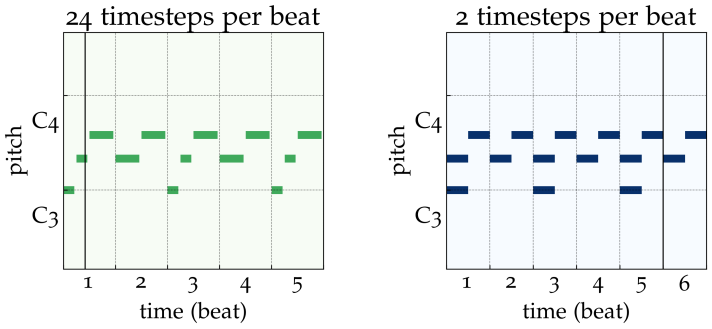


Figure 4.3: Input and output windows

System resolution and downsampling

According to what is described in section 3.1.2 regarding the beat resolution of piano rolls, a good choice would be to set it to 24, i.e. twenty-four timesteps per beat. In such case, however, a window of 10 timesteps would be equal to less than half a beat, resulting in a very short temporal range for predictions, as shown in fig. 4.4a. Some postprocessing can be made in order to improve quality and usefulness of the predictions: by applying a downsampling process with a factor of 12, the new piano roll would contain 2 timesteps per beat which means that 10 timesteps are now equivalent to 5 beats, as shown in fig. 4.4b, which in a common $\frac{4}{4}$ time signature is equivalent to a whole measure and a beat.

After the downsampling process, the system resolution becomes equal to one *quaver* (♪), meaning that predicted notes have a minimum duration of $1/8$ of a semibreve (recall table 3.2). This process clearly causes a loss in resolution since multiple fast notes are aggregated together in the same timestep, so many ornaments and flourishes are not preserved; however, since the model should be able generate music that sounds musically pleasant and coherent, there is no particular need to deal with such fast notes. The only thing that has to be considered is that an inverse process, called *upsampling*, has to be performed to revert the piano roll of predictions into the original shape in order to fit the original track.



(a) Original **(b)** Downsampled

Figure 4.4: Piano roll downsampling. The vertical lines delimit the 10th timesteps in the respective pian rolls.

4.2 NEURAL NETWORKS MODELS

The machine learning models that have been trained belong to two different categories: feedforward and recurrent networks. Both models are implemented with the help of Keras Sequential Class [20], which allows to easily stack different layers to build the required model. Some of the layers used in this framework are reported below:

DENSE Fully connected layer where each neuron has a connection with all neurons of the previous layer.

FLATTEN Layer that squeezes the dimensions of the previous layer by outputting a 1-D reshaped vector.

INPUT Symbolic layer which defines the input shape of the model.

SIMPLERNN Implementation of the RNN cell.

GRU Implementation of the GRU cell.

LSTM Implementation of the LSTM cell.

The chosen loss function is the *binary cross-entropy*, which is the cross-entropy reported in eq. (4) adapted to the case where each note can assume only two values (0 and 1), while the activation function at the output is the *sigmoid*, since it suits the multi-label classification problem allowing different classes (pitches) to be active at the same time.

The optimisation algorithm used for gradient descent is Adam, with a learning rate $\eta = 0.001$.

4.2.1 Feedforward model

The Keras Sequential implementation of the feedforward model is represented in listing 1. The first Dense layer receives as input an instance that is made of one or several timesteps, each of them including all pitches in the piano keyboard. The Flatten layer reshapes the output from the previous layer to squeeze the 2-D matrix into a 1-D vector. The last Dense layer returns a vector which is as long as the number of notes of the keyboard multiplied by the number of future timesteps that have to be predicted, in this specific case 64 notes multiplied by 10 timesteps.

Listing 1: Feedforward model

```
Layer (type)          Output Shape       Param #
=================================================================
dense (Dense)         (None, 10, 32)     2080
_________________________________________________________________
flatten (Flatten)     (None, 320)        0
_________________________________________________________________
dense_1 (Dense)       (None, 32)         10272
_________________________________________________________________
Output (Dense)        (None, 640)        21120
=================================================================
Total params: 33,472
Trainable params: 33,472
Non-trainable params: 0
```

4.2.2 Recurrent model

The different implementations of the recurrent model are represented in listings 2 to 4. The first actual layer, which is not listed in the summaries, is an *Input* layer by which the input shape is defined in order to implement the recurrent cell as the first processing layer. Such recurrent layer has the

parameters `return_sequences` and `stateful` set both to `True`, meaning respectively that the sequence of predictions is returned after each loop inside the network and the state of the system is not reset among batches. This makes it possible to keep information about past data among the different iterations.

These models have also a larger number of parameters compared with the feedforward model; this would easily causes overfitting during training, but this issue can be mitigated by training the models on small number of epochs or, after the training, by resuming the weights from the point when the models were performing best.

Listing 2: SimpleRNN model

```
Layer (type)          Output Shape        Param #
=================================================================
simple_rnn (SimpleRNN) (64, None, 128)    98432
_________________________________________________________________
dense (Dense)         (64, None, 64)      8256
_________________________________________________________________
Output (Dense)        (64, None, 640)     41600
=================================================================
Total params: 148,288
Trainable params: 148,288
Non-trainable params: 0
```

Listing 3: GRU model

```
Layer (type)          Output Shape        Param #
=================================================================
gru (GRU)             (64, None, 128)     295680
_________________________________________________________________
dense (Dense)         (64, None, 64)      8256
_________________________________________________________________
Output (Dense)        (64, None, 640)     41600
=================================================================
Total params: 345,536
Trainable params: 345,536
Non-trainable params: 0
```

Listing 4: LSTM model

```
Layer (type)          Output Shape        Param #
=================================================================
lstm (LSTM)           (64, None, 128)     393728
_________________________________________________________________
dense (Dense)         (64, None, 64)      8256
_________________________________________________________________
Output (Dense)        (64, None, 640)     41600
=================================================================
Total params: 443,584
Trainable params: 443,584
Non-trainable params: 0
```

It can be seen that among all the proposed models, the one featuring the LSTM cell has the largest number of parameters, followed by the Gated Recurrent Unit (GRU) cell: that is because, as described in section 3.3.5, these

recurrent cells are very complex when compared, for example, with a SimpleRNN cell.

4.3 BASELINE MODEL

The baseline is a simple model which is supposed to perform the same task of the machine learning model, but it does not require a training phase. Its behaviour is hard coded and always follow some simple rules. Some possible ways to implement a baseline model for the music prediction task are listed below:

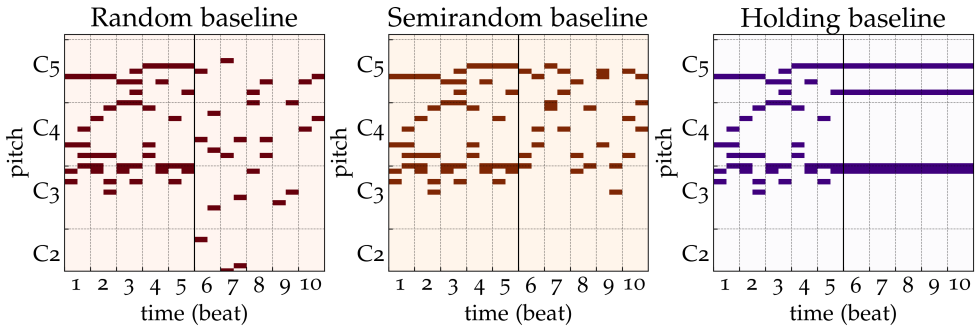
RANDOM The predictions are generated in a random way, spanning the entire set of notes, without constraints about tempo or tonality.

SEMIRANDOM The predictions are generated randomly from a set of notes taken from some previous timesteps, in order to keep a certain coherence in the tonality.

HOLDING The predictions consist in the repetition of the last status of the system. It can be seen has a model that plays a *frozen* status of the music stream, from the timestep when an error has occurred.

It is intuitive that a *random* approach can produce far from satisfactory results, instead a *semirandom* baseline could produce something that sounds remotely pleasant, at least regarding the tonality of the song. The holding baseline however can produce something that is somewhat more imaginable since it represents a suspended status of the system when it freezes. These baseline model behaviours are shown in fig. 4.5: the piano roll on the first five beats is the original track; at beat 6 an error occurs and the next five beats are made up by the baseline model.

For this thesis, the artificial neural networks models have been compared with the baseline that holds the last status of the system.



(a) Random baseline **(b)** Semirandom baseline **(c)** Holding baseline

Figure 4.5: Different baseline models for prediction tasks

4.4 PREDICTION POSTPROCESSING

4.4.1 Thresholding

After training an artificial neural network model, its output during the test phase do not necessary respect the desired form, meaning that if a model is trained to output binary values, it usually outputs values that are very close to a binary form, e.g. 0.017 and 0.989, but not actual integer binary values such as 0 and 1. This means that a post-processing of the output via a thresholding phase has to be performed:

SIMPLE THRESHOLD For binary-classification problems, an intuitive threshold can be set to 0.5, meaning that every float value between 0 and 1 is converted to the nearest integer value, which in the end produce a binary output. This is for sure a really fast way to get the output from the model in the desired form; however it does not necessary fits all problems like the one proposed in this thesis. In many cases the threshold at the exact half of the sides does not produce satisfying results and has to be lowered or increased to prevent unbalanced classifications.

VALIDATION THRESHOLD A similar but more advanced solution can be implemented by using an algorithm which computes the converted output for several different threshold values. This process is usually done on the validation set. For each threshold value, an accuracy value is evaluated by comparing the thresholded output with the labels: the one that provides the highest accuracy can be chosen as threshold value.

RANKED THRESHOLD A different approach, which fits multi-label classification problems, consists in sorting the samples in each output instance by their probability, then choosing a number n of those samples and assigning them the value 1. By doing this, every output instance would have a predefined number of positive samples, ignoring the concept of threshold applied to values.

For this thesis, the thresholding phase has been made by using the *ranked threshold* method.

4.4.2 Reshaping

Since the neural network models for every input instance return an output that consists of a single 1-D vector, such vector has to be reshaped in order to recreate the 2-D window that fits the piano roll representation. This can be done by simply considering all timesteps in the window as they were put in sequence one after the other in the 1-D vector. The choice of using the vector over a matrix as output for the neural networks is due to practical reasons since Dense layers return for each instance a 1-D output.

5 NUMERICAL ASSESSMENT

In this chapter, the performance evaluation of the proposed framework is presented. In particular, several experiments are performed, including prediction of single timesteps and predictions of multiple timesteps using both feedforward and recurrent neural networks models. The performance of the models is evaluated using as reference the *ground truth* which is the piano roll of the real track; therefore a *supervised* learning approach is used.

5.1 EVALUATION METRICS

Different metrics are considered: precision, recall and F1 score are used for the first experiment to evaluate the learning capability of the network in general; the AUC-ROC is then used to compare the model with the baseline model and see the difference in prediction quality.

At the end, a metric based on consonance perception is considered to compare the model with the baseline from a musical point of view.

5.1.1 Precision, recall and F1 score

Precision and recall are common metrics in binary classification since they can simply describe how well a system retrieves relevant results. Precision represents the ratio between what the model predicts correctly and what the model predicts; recall represents the ratio between what the model predicts correctly and what the labels actually are. Both metrics are mathematically computable with the help of positives and negatives related variables. A graphical representation of both metrics is shown on fig. 5.1.

F1 score can instead be evaluated by knowing in advance precision and recall values. All formulas are reported below.

TRUE POSITIVES (TP) Positive values that are correctly classified.

FALSE POSITIVES (FP) Negative values that are classified as positives.

FALSE NEGATIVES (FN) Positive values that are classified as negatives.

$$\text{precision} = p = \frac{|\{\text{correct predictions}\}| \cap |\{\text{all predictions}\}|}{|\{\text{all predictions}\}|} = \frac{TP}{TP + FP}$$

$$\text{recall} = r = \frac{|\{\text{correct predictions}\}| \cap |\{\text{all predictions}\}|}{|\{\text{correct predictions}\}|} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = 2\,\frac{p \cdot r}{p + r}$$

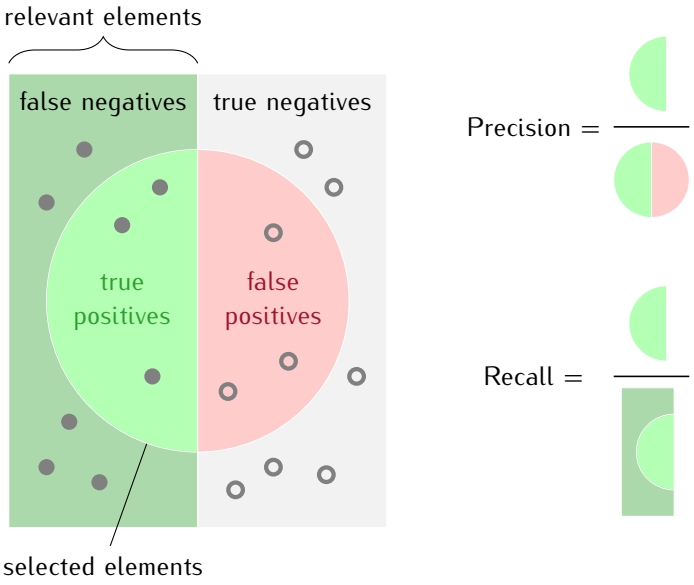


Figure 5.1: Precision and recall: graphical representation. Precision corresponds to how many selected items are relevant. Recall corresponds to how many relevant items are selected.

5.1.2 AUC-ROC

The Receiver Operating Characteristic Curve (ROC) curve is a graphical plot which illustrates the diagnostic ability of a binary classifier system for different discrimination thresholds. It is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.

The Area Under the Curve (AUC) metric corresponds to the area that is subtended — in this case — by the ROC curve and it is equal to a number which is useful to evaluate the bounty of the model. Its maximum value is 1, meaning that positive values are always retrieved as such and there are no false positives. A value equal to 0.5 generally means that the classifier is not able to discern between positive and negative cases, so it behaves like it is taking random decisions.

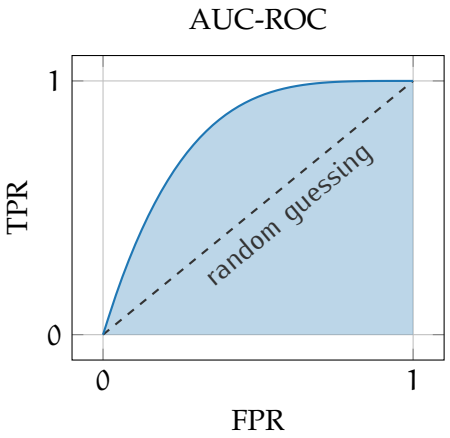


Figure 5.2: AUC-ROC metric: graphical representation. The curve represents the ROC, the highlighted area represents the AUC value. An AUC value of 0.5 corresponds to a classifier that takes random decisions.

5.1.3 Consonance perception

The consonance perception metric is based on the width of stability for two-tone intervals proposed by Hermann von Helmholtz in [12]; in particular, the values in table 5.1 that come from reference [11] are taken into consideration. These values reflect how much a music interval between two pitches is considered consonant or dissonant to human hearing. This can be exploited to evaluate how much the model predictions are considered coherent or not with respect to the ground truth.

The difference between this kind of metric and the AUC-ROC metric is that in the latter the errors in predictions are rated so that the model is penalised more when it generates dissonant notes with respect to the ground truth.

The idea is taking a note p_t from the real track played at timestep t and finding the nearest predicted note by the model at the same timestep. The distance between these pitches is converted to a score value according to table 5.1. This process is repeated for every played note in the track and the average score is in the end computed. Equations (6) and (7) represent the formula used for the metric, where:

- N_t: total number of timesteps included in the track
- N_{p_t}: total number of ground truth notes active at timestep t
- p_t: ground truth note belonging to timestep t
- a_t: prediction note belonging to timestep t

$$\text{score}_{p_t} = f\left(\min_{a_t} |p_t - a_t|\right) \tag{6}$$

$$\text{score} = \frac{1}{0.075} \cdot \frac{1}{N_t} \left(\sum_{t=0}^{N_t} \frac{1}{N_{p_t}} \cdot \sum_{p=0}^{N_{p_t}} \text{score}_{p_t} \right) \tag{7}$$

In this way, eq. (6) evaluates the score associated to a single note of the ground truth, which represent how well it is predicted by the model. Equation (7) is the average score over all notes and all timesteps, converted in percentage: the coefficient $1/0.075$ is a normalising constant to convert the score into the range $[0, 1]$.

It must be said that this metric does not consider the tonality of the song since this information is not included in the dataset and cannot be explicitly used during training. For this reason, the concept of consonance is relative only to the intervals between the notes and not the tonality.

5.2 PREDICTION OF ONE TIMESTEP

The simplest experiment consists in training a model that is able to predict a set of notes at a particular time instant, given the set of notes at the previous timestep, i.e. using a window size equal to 1. Because of this, the number of units of the last layer of the network is equal to the keyboard size, in this case a standard 88-key piano keyboard. Different metrics are considered:

Interval	Distance (semitones)	Value $f(D)$
Unison	0	0.075
Octave	12	0.023
Fifth	7	0.022
Fourth	5	0.012
Major sixth	9	0.010
Major third	4	0.010
Minor third	3	0.010
Minor sixth	8	0.007
Major second	2	0.006
Major seventh	11	0.005
Minor seventh	10	0.003
Minor second	1	0
Tritone	6	0

Table 5.1: Values of consonance perception

F1 score, precision and recall. The beat resolution of MIDI files is set to 24, meaning that every beat of the song is represented by 24 timesteps. In addition, no downsampling process is performed.

The dataset used is Classical Piano which contains 87 training files, 12 validation files and 25 test files, divided in such a way that songs from the same composers are spread onto different subsets.

The results for loss function and metrics are reported in table 5.2, from which it is visible that the model achieves a very low binary cross-entropy and very high scores for the other metrics, meaning that the predictions can be considered as accurate. Those results are also reported in figs. 5.3 and 5.4, where all curves reach a convergence in few epochs.

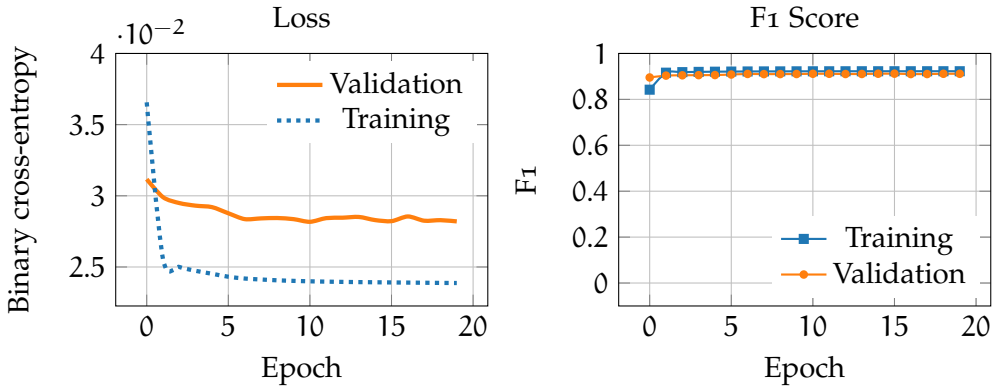
Hence, the model is very good at predicting one timestep in the future because it is very likely that notes that are active at timestep t are also active at timestep $t+1$, due to the high number of repeated occurrences per note, which is the result of not having applied the downsampling process.

Dataset	Loss	F1 Score	Precision	Recall
Training	0.024	0.889	0.927	0.924
Validation	0.028	0.890	0.925	0.920
Test	0.024	0.908	0.925	0.919

Table 5.2: Prediction of one timesteps: metrics

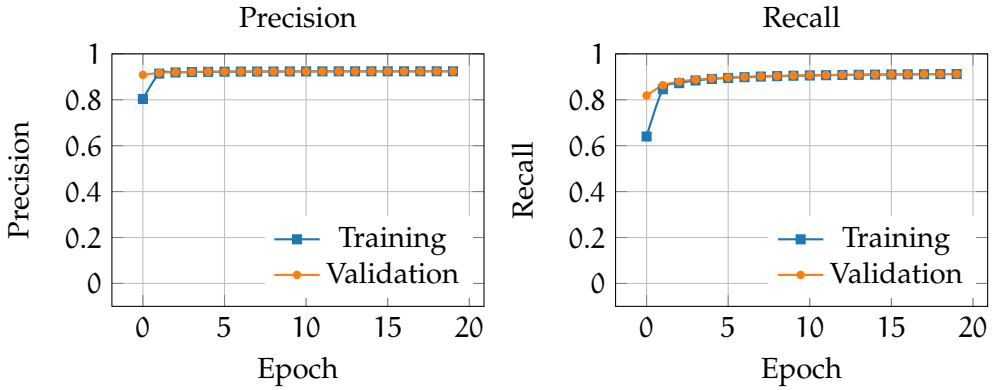
DIFFERENT BATCH SIZES The experiment is then repeated with different batch sizes, to see how such parameter affects the way the network learns. Note that batch size are always kept a power of 2 in order to fit the storage size of processing units.

Results are reported in table 5.3 from which it is visible that increasing batch size significantly reduces computational time; in spite of this, the algorithm cannot reach convergence if the batch size is too large. This behaviour



(a) Loss **(b)** F1 score

Figure 5.3: Loss and F1 Score; 20 epochs; batch size equal to 64



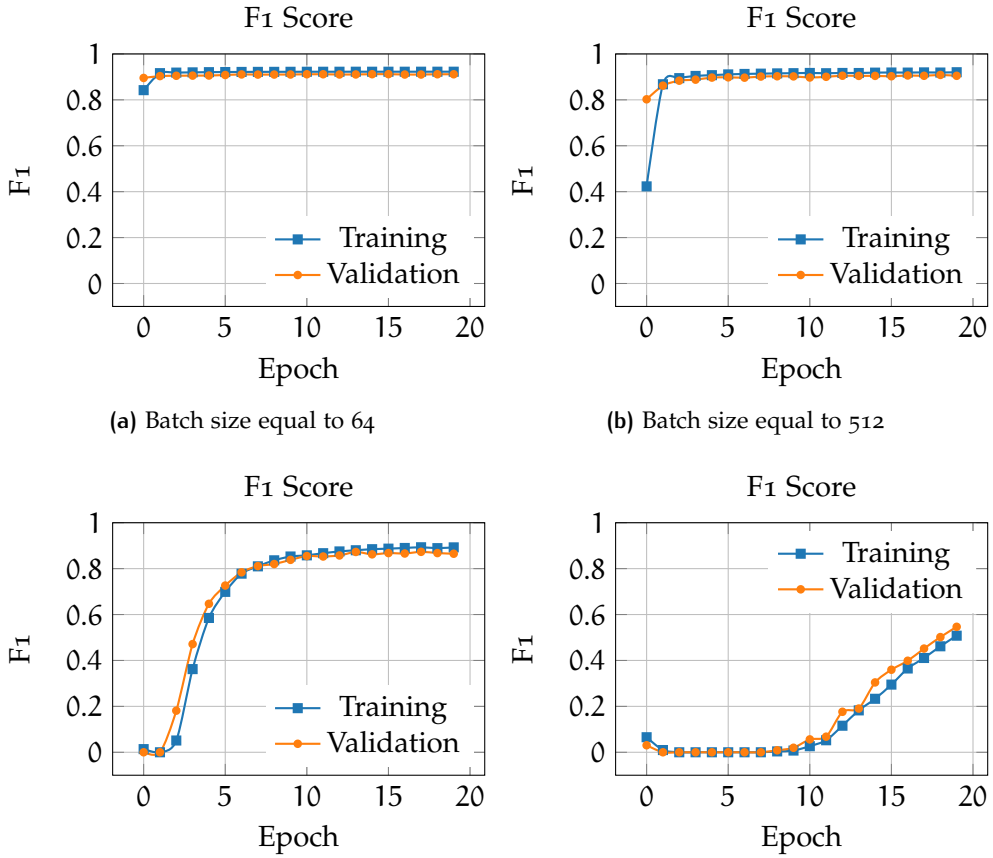
(a) Precision **(b)** Recall

Figure 5.4: Precision and Recall; 20 epochs; batch size equal to 64

is evident in fig. 5.5 from which it can be supposed that increasing the batch size actually slows down training, since the number of epochs have to be increased in order to compensate the slower convergence, even though the computational times per single epoch are reduced. For this reason, a batch size equal to 64 has been kept for all experiments.

Batch size	Training time (s)	Loss	F1 Score	Precision	Recall
64	1205.72	0.028	0.890	0.925	0.920
512	156.03	0.030	0.896	0.925	0.915
4096	21.69	0.037	0.864	0.923	0.847
20 480	7.12	0.077	0.502	0.870	0.340

Table 5.3: Comparison among different batch sizes; 20 epochs; metrics for validation set



(a) Batch size equal to 64

(b) Batch size equal to 512

(c) Batch size equal to 4096

(d) Batch size equal to 20480

Figure 5.5: F1 score over epochs for different batch sizes. The higher the batch size, the higher the amount of epochs required to reach a convergence.

5.3 PREDICTION OF MULTIPLE TIMESTEPS

The experiment is then extended to the prediction of multiple timesteps, given a windows of past timesteps. In particular, the model is trained to take windows of 10 past timesteps $[t_{n-10}, t_{n-1}]$ as input to predict the next 10 future timesteps $[t_n, t_{n+9}]$ assuming that an error occurs at t_n (recall fig. 4.3). To achieve this, the number of units of the last layer of the model is multiplied by a factor of 10 with respect to the experiment in section 5.2, since the number of notes that have to be predicted is ten time larger, which also increases the number of parameters that have to be trained.

For this experiment, the AUC-ROC metric is considered and it is evaluated for every pitch at every timestep; this allows to build a heatmap which represents the general behaviour of the model in predicting notes. The model is also compared with the baseline model that repeats the last status of the system before the error.

The results are reported in fig. 5.6: every row represents a pitch of the 88-key piano keyboard; every column represents a timestep in the future which has to be predicted; white rows correspond to notes that are never played in the test set, so they are not considered by the metric. It is visible that the model behaves better than the baseline since the graph tends to lighter colours. Recall that an AUC-ROC value of 0.5 means that the model is pulling at random if a note is played or not, whereas a value of 1 means that the model knows perfectly which notes will be played and which notes will not. From the heatmap it can be inferred that the timesteps closer to the first one are easier to predict, while the furthest are more difficult; moreover the central range of pitches seems to be more difficult to model with respect to lower and higher ends; this could be because lower pitches usually belong to basslines which have a more predictable pattern, whereas higher ones are rarely played.

A different representation is depicted in fig. 5.7, which shows the AUC-ROC averaged over the notes for every future timestep. As said before, it is reasonable that the timesteps very near in the future are easier to predict with respect to further ones. For this reason, the AUC-ROC values for both model and baseline decreases with future timesteps. The baseline model is more affected since it just provides repetitions of the last timestep in the input.

5.4 DOWNSAMPLING AND DATA AUGMENTATION

5.4.1 Data preprocessing

The issue that emerges from the previous experiment is that, even if the number of predicted timesteps is equal to 10, those predicted notes are not so useful in a pratical case since they would correspond to less than half a beat (recall fig. 4.4). There are some countermeasures that have been implemented to help the training phase and make the model prediction more meaningful. To make predictions further in the future, without increasing the model complexity, the downsampling process described in section 4.1.2

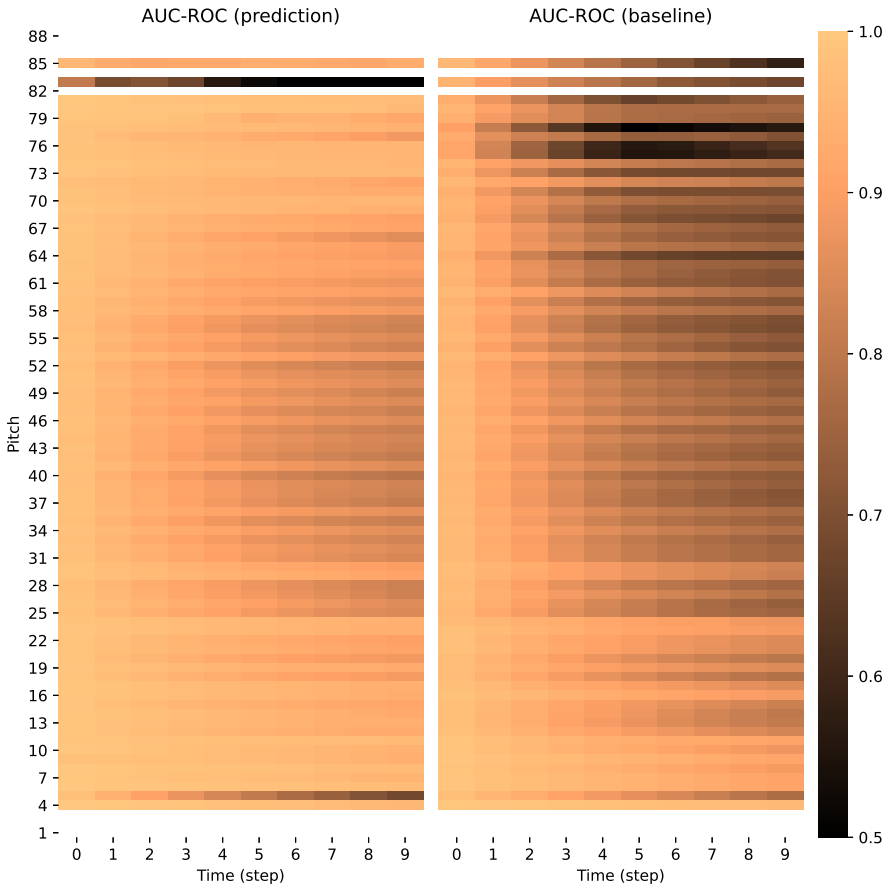


Figure 5.6: Prediction of multiple timesteps: AUC-ROC heatmap. White cells represent NaN values. Pitch no. 49 is A_4 (440 Hz)

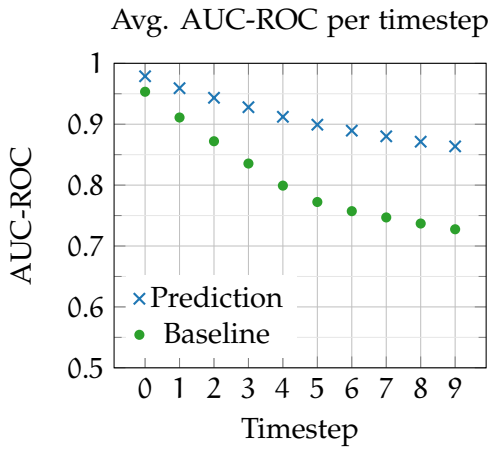


Figure 5.7: Prediction of multiple timesteps: mean AUC-ROC averaged over all notes for different predicted timesteps. Timestep no. 0 is the prediction given the very last status of the system, so it is supposed to be the most accurate one.

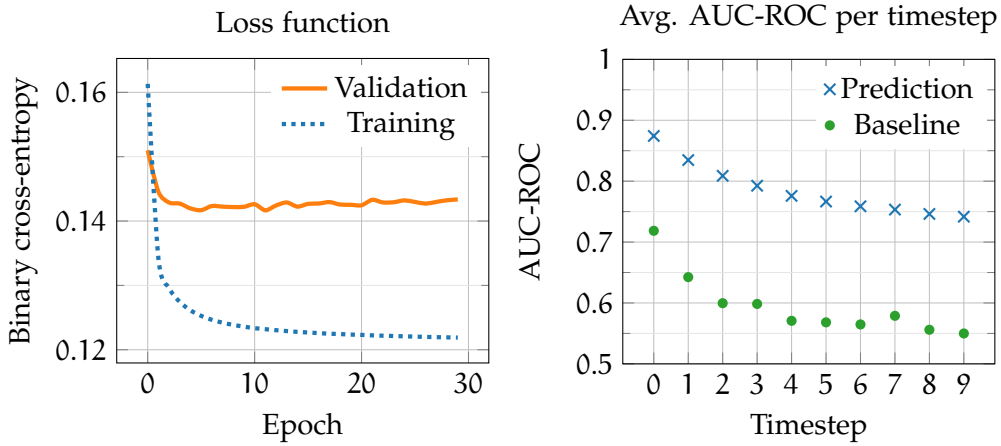
is applied. Such process consists in aggregating many timesteps into a single one, preserving all notes that are played. In this case the procedure is the following:

1. Every MIDI file is converted with a *beat resolution* equal to 24, which means 24 timesteps per beats in the piano roll representation.
2. The piano roll is then downsampled, with a factor of 12, to 2 timesteps per beat in order to have one timestep per quaver.
3. Every instance is built with two windows made of 10 timesteps each, meaning 10 timesteps to predict 10 timesteps. Ten timesteps are now equal to five beats, more than a complete measure in a common $\frac{4}{4}$ time signature.

Data augmentation is also performed on the training dataset in order to transpose every batch to a random interval of semitones in the range $[-5, +5]$. This process largely increases the initial training set size and helps to prevent overfitting and to reduce the dependency between musical patterns and absolute pitches.

The model is then trained on such modified dataset. Figure 5.8a shows the loss function from which it is visible that the model is learning and there is no overfitting, while fig. 5.8b shows the average AUC-ROC which confirms that the neural network model predictions are always better than the simple baseline ones. This is also visible in fig. 5.9 where the heatmap of the baseline is much darker than the one referring to the model.

Comparing the AUC-ROC curve of fig. 5.8b with the previous curve obtained from non-downsampled data in fig. 5.7, it is visible that the new dataset apparently results in worse predictions; this is because now the predicted timesteps are covering five beats instead of half a beat, and this makes accurate predictions harder to be achieved. In spite of this, the predictions which are now more extended in the future are also more meaningful.



(a) Binary cross-entropy

(b) Avg. AUC-ROC

Figure 5.8: Downsampling and data augmentation: loss function and AUC-ROC

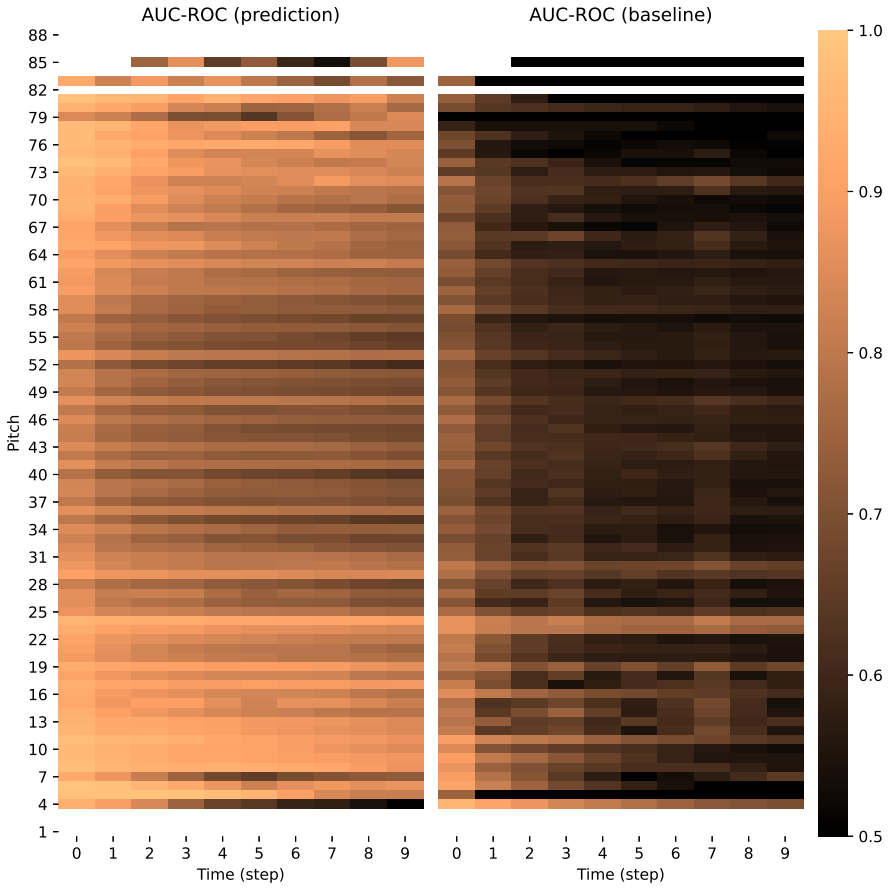


Figure 5.9: Downsampling and data augmentation: AUC-ROC heatmap

5.4.2 Model predictions

This section shows some model predictions based on the feedforward model trained on the Classical Piano dataset and visualised through a piano roll representation. The first part of the piano roll shows the real track which is played through the music stream; the second part represents the gap that is filled by model predictions.

Figure 5.10 shows a simple pattern on the C major chord. The error occurs at timestep 25; therefore the model takes as input timesteps 20–24 and returns a prediction of timesteps 25–30. On the piano roll patched by the feedforward model, it is visible that the predicted notes include only the pitches played in the pattern and such pattern is somehow maintained during the gap. On the contrary, the baseline model just holds the single note that was active before the interruption.

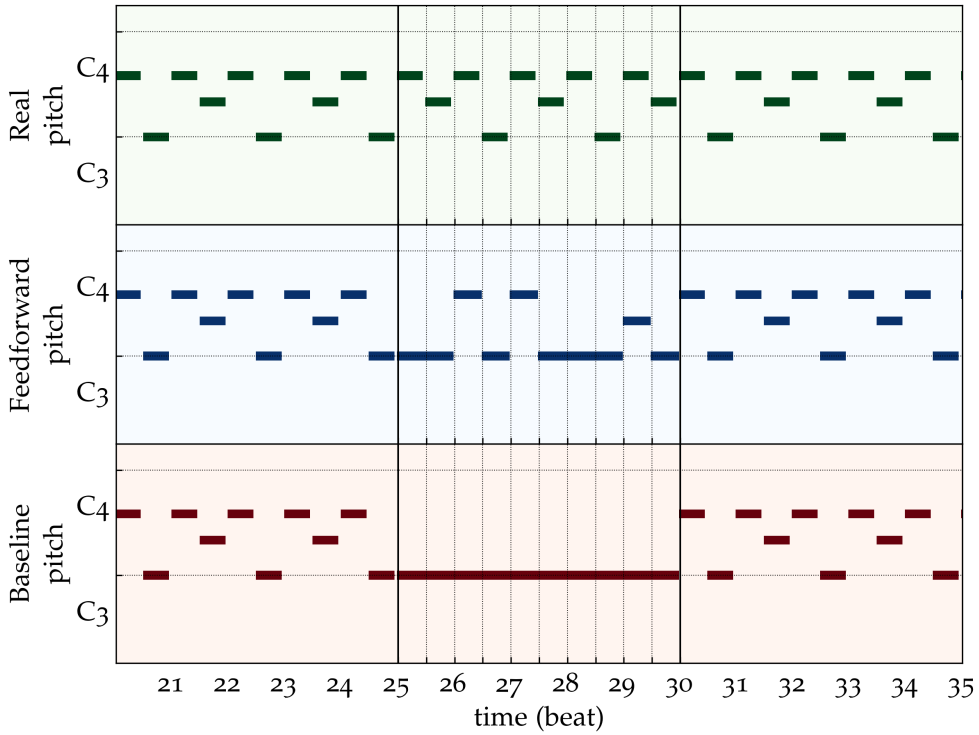


Figure 5.10: Predictions on a simple pattern on C major chord

Figure 5.11 shows an extract from the Minuet in G major (BWV Ahn 16) by Johann Sebastian Bach. It is musically more complex with respect to the pattern in C major proposed before. In spite of this, the feedforward model manages to keep track of the tonality of the piece and fill the gap with notes that are coherent with the melody.

Figure 5.12 shows a piece taken from the test set of Classical Piano. The predictions of feedforward model cannot be considered as 'accurate' with respect to the ground truth in an absolute way; however, comparing them with the ones made by baseline model, they are acoustically more pleasant and coherent, which is also evident by listening to the audio files.

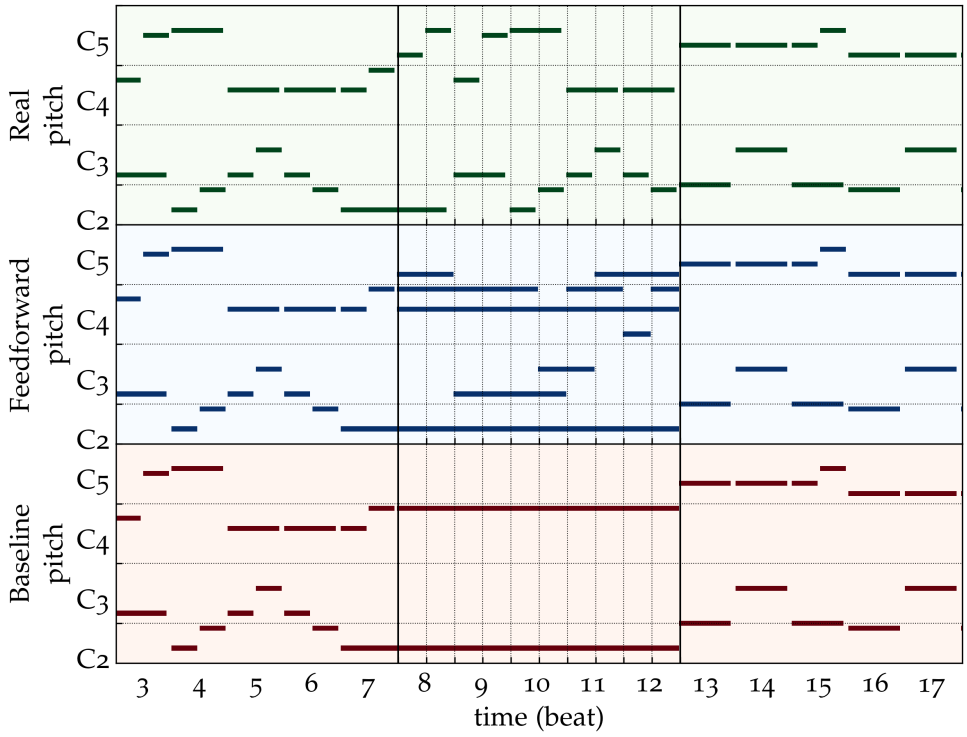


Figure 5.11: Predictions on a Minuet in G major by Johann Sebastian Bach

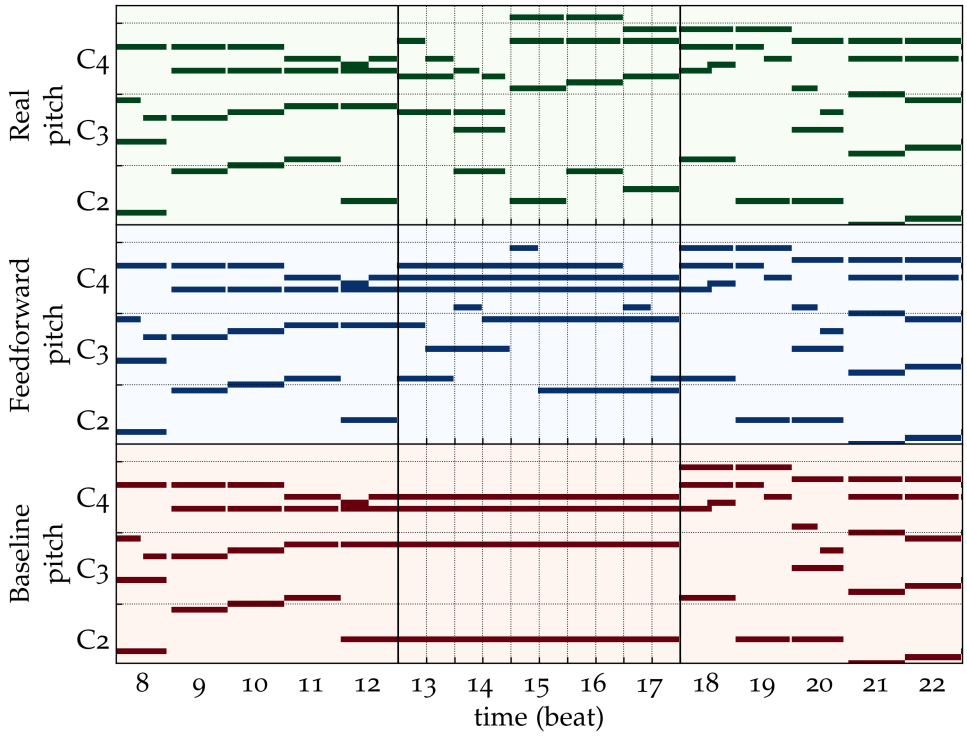


Figure 5.12: Predictions on Fantasien Op. 116 by Johannes Brahms

5.5 DIFFERENT KEYBOARD SIZES AND DATASETS

5.5.1 Keyboard sizes

Another parameter which can be changed is the number of notes included in the piano rolls used to train the model. For this experiment, the model has been trained using different keyboard sizes. Considering that the majority of played notes are in general included in the middle octaves of an 88-key keyboard, the following reduced sizes have been tested:

88 KEYS From A_0 to C_8 (standard Grand Piano keyboard)

64 KEYS From A_1 to C_7

49 KEYS From C_2 to C_6

From fig. 5.13 it is visible that reducing the number of notes does increase the loss function; this happens because the loss is evaluated with the binary cross-entropy which takes into consideration not only correctly predicted notes, but also how many silent notes are predicted as silence. Supposing that the larger the keyboard size, the higher is the number of notes that are never played, this behaviour should be expected.

However, fig. 5.14 shows that the AUC-ROC metric is almost the same even if the total number of notes is reduced. This means that the model learns to predict the notes in the correct way, disregarding the number of always-silent notes. In any case, a piano roll made of 64 pitches seems to provide slightly better results.

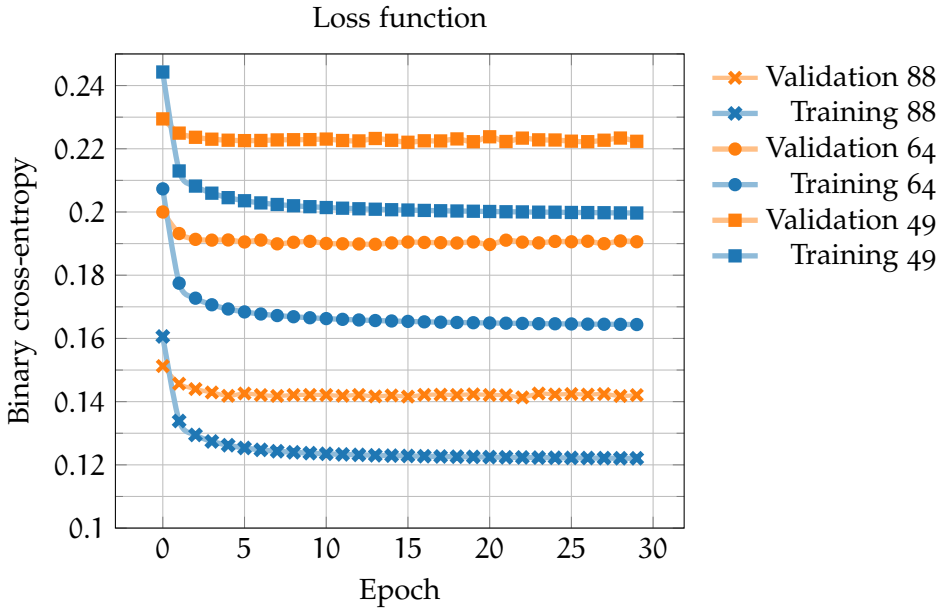


Figure 5.13: Different keyboard sizes comparison: loss function

5.5.2 Datasets

The model is then trained on the three different datasets described in section 3.1.3 to find out differences in performance among those.

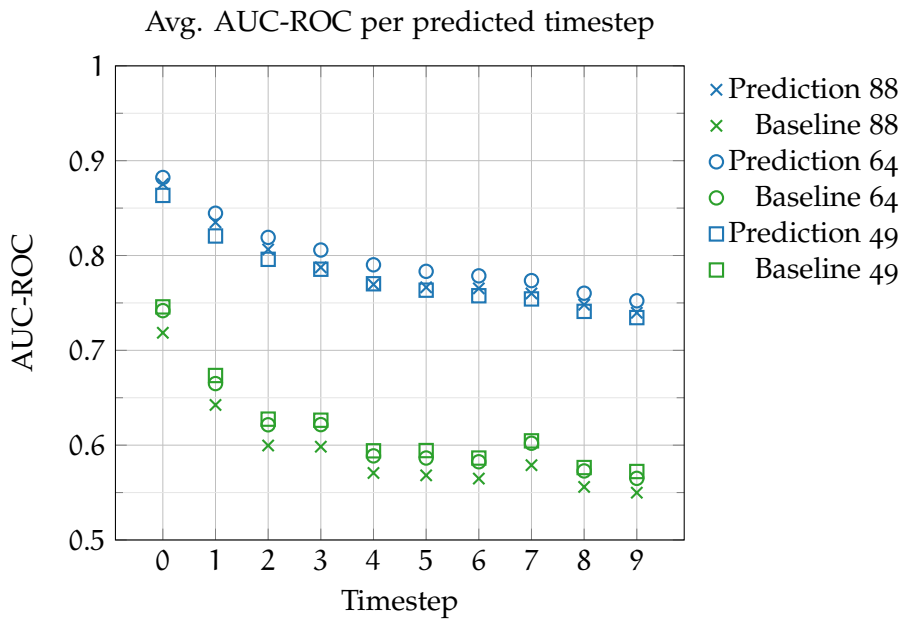


Figure 5.14: Different keyboard sizes comparison: AUC-ROC

Regarding the loss functions shown in fig. 5.15, it is visible that the lowest values are reached with Nottingham database, followed by J. S. B. Chorales and then Classical Piano. It is also noticeable that the validation losses on Nottingham and J. S. B. Chorales are very close to the respective training losses; this means that these datasets are relative easy to model with respect to Classical piano whose validation and training losses are more distant.

Regarding the AUC-ROC metric represented fig. 5.16, it seems that the one that provides best results is relative to chorales by J. S. Bach, especially by looking at the first and last predicted timesteps, whereas the most difficult to predict is the Classical Piano, probably due to its musical inhomogeneity.

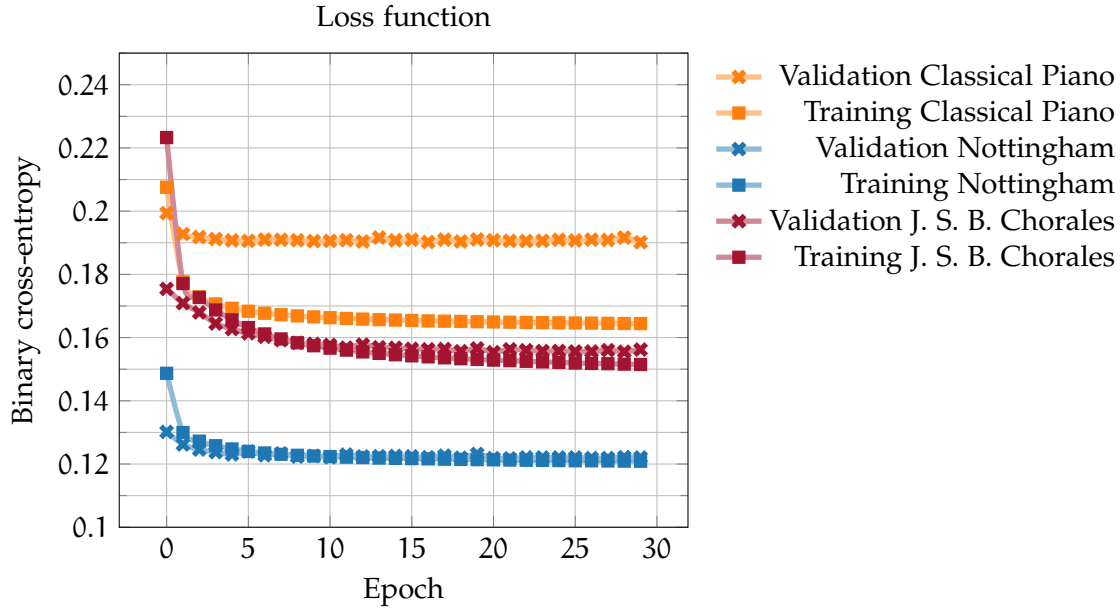


Figure 5.15: Different datasets comparison: loss function

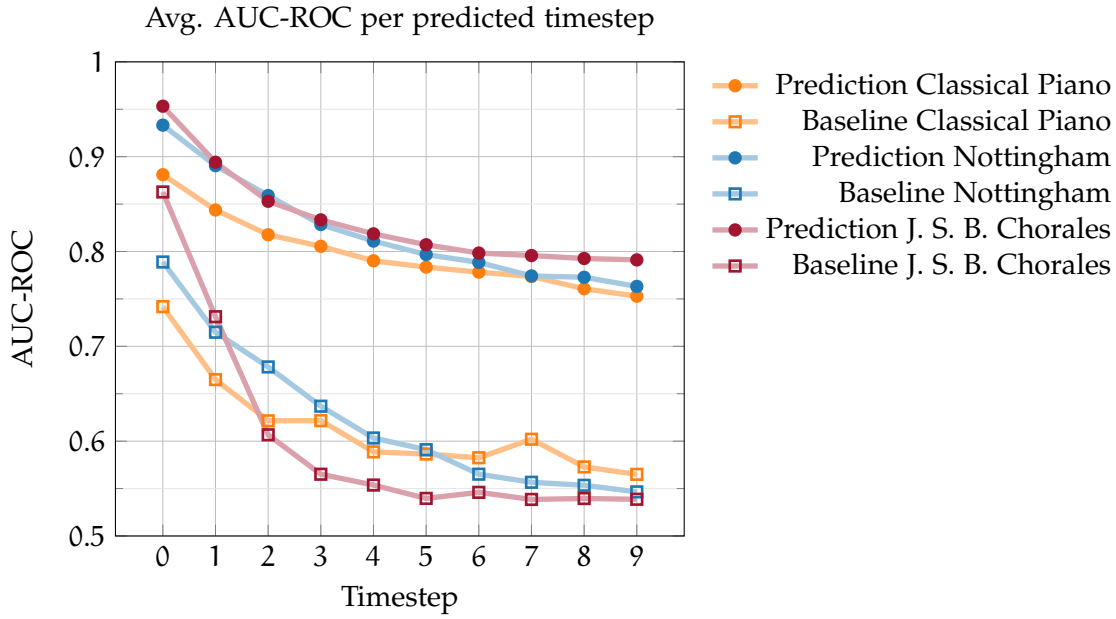


Figure 5.16: Different datasets sizes comparison: AUC-ROC

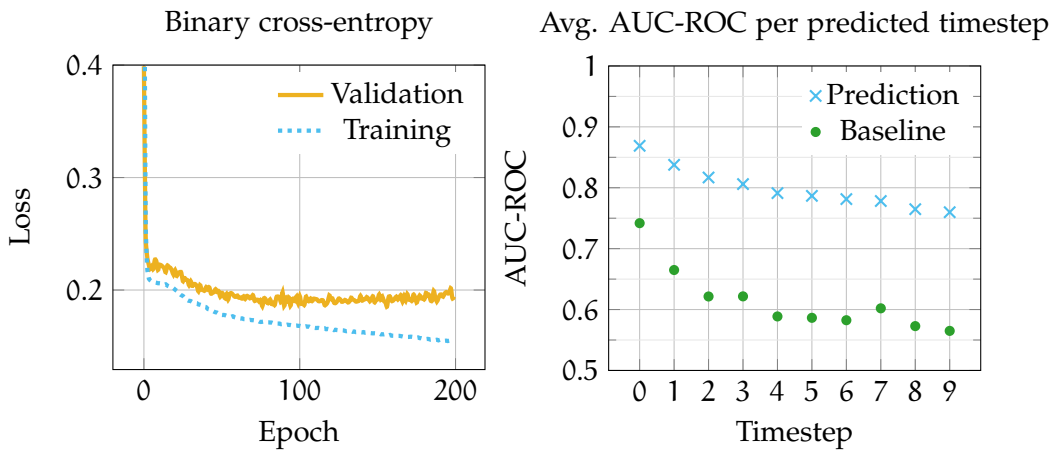
5.6 RECURRENT MODELS COMPARISON

The recurrent models are made of a recurrent layer instead of a Dense layers with respect to the feedforward one, and they feature a significantly larger number of trainable parameters. Because of that, they are more inclined to overfitting, which has to been mitigated by limiting the number of epochs used for training.

Figure 5.17 shows as example the loss function for the recurrent model which includes the LSTM layer trained on the Classical Piano dataset: it is visible that the validation curve begins to rise after 100 epochs, i.e. when the model starts to overfit. In any case, using the weights corresponding to the lowest validation loss, which during training are saved as *checkpoints*, it is possible to test the model using that particular state at which it was performing best. Results on test set regarding AUC-ROC metric are shown in fig. 5.17b, where it can be seen that the recurrent model is clearly capable to beat the baseline model on every future timestep.

The three different recurrent models are trained on the Classical Piano dataset to find the differences in performance which are shown in fig. 5.18. It is visible that all models seem to achieve similar results in term of AUC-ROC; however the LSTM model performs always better than the others: its recurrent cells, in fact, process useful information in a more efficient way.

The recurrent model featuring the LSTM is then trained on all datasets. The results are shown in fig. 5.19, the behaviour is very similar to the one of the feedfoward model (recall fig. 5.16) in which the baseline model is always outperformed. However it can be noticed that this time the Nottingham database seems to give better performance than the J. S. B. Chorales, while for the feeforward model this behaviour was the other way around.



(a) Binary cross-entropy

(b) AUC-ROC

Figure 5.17: LSTM model: loss function and AUC-ROC on Classical Piano database

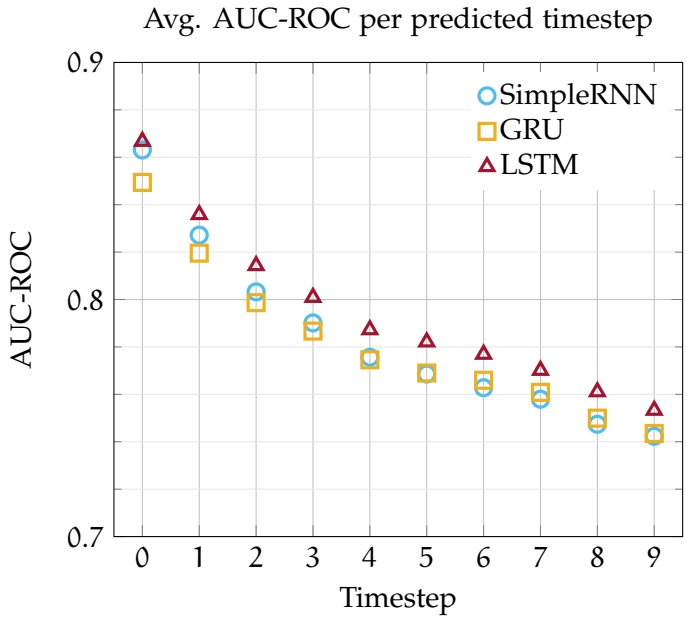


Figure 5.18: AUC-ROC for different recurrent models on Classical Piano dataset

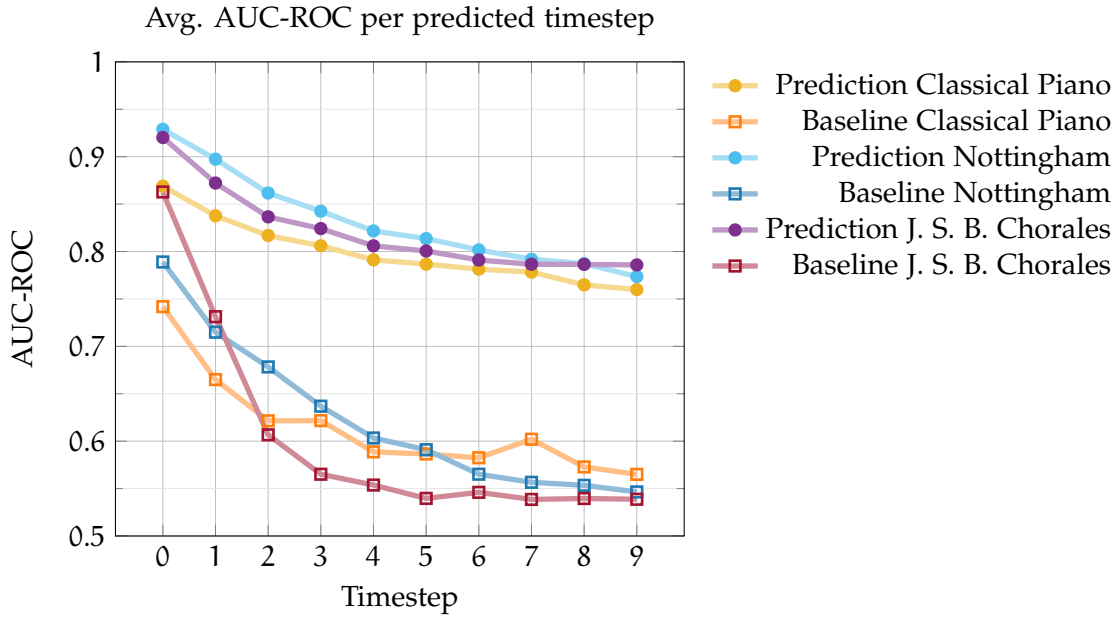


Figure 5.19: LSTM model: AUC-ROC on different datasets

5.7 CONSONANCE PERCEPTION EVALUATION

This section shows the performance of the two neural networks model and the baseline using the metric based on consonance perception described in section 5.1.3. Regarding the recurrent model, only the one including the LSTM cell is considered since it is the one that performs best according to fig. 5.18. Performance is then evaluated on all datasets.

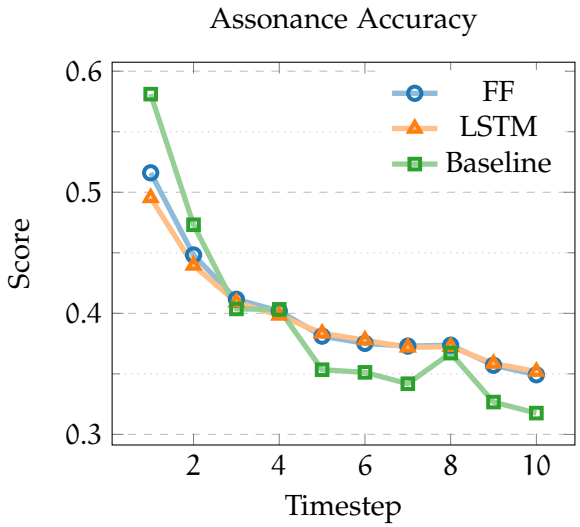
Performance on Classical Piano dataset shown in fig. 5.20a is very similar for the two artificial neural networks models, while the baseline seems to perform better on the very first timesteps.

On the Nottingham dataset, shown in fig. 5.20b, results are always in favour of neural network models which always beat the baseline with a difference that increases for the furthest timesteps.

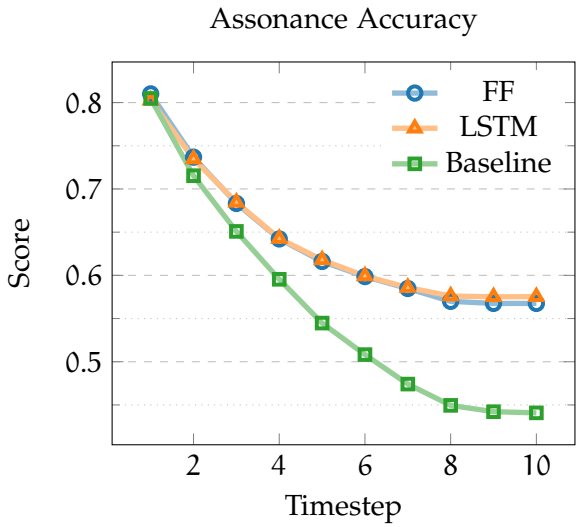
Regarding the J. S. B. Chorales dataset, the feedforward model outperforms all the other models. It is also noticeable that the LSTM model performs worse than the baseline for the first two timesteps.

5.8 EXAMPLES OF PREDICTED PIANO ROLLS

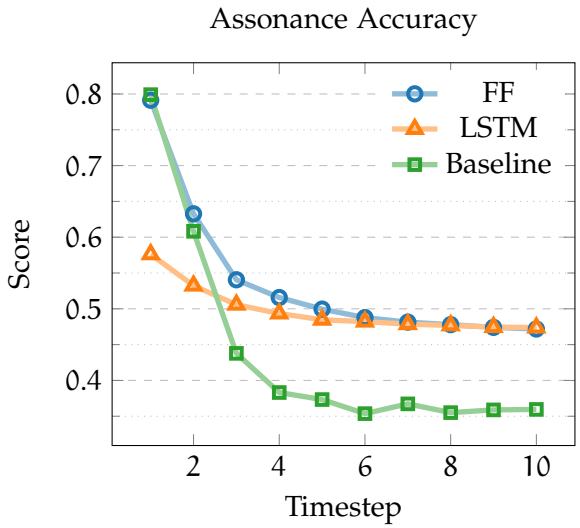
This section shows some examples of test songs that have been patched by feedforward, recurrent (LSTM) and baseline models during simulated errors. All piano rolls show on the left a window of 5 beats that corresponds to the real track given as input to the models, a middle window of 5 beats where a gap in the playback is supposed to be and which is filled by the output of the models, and a last window of the resumed playback corresponding to the real track.



(a) Classical Piano dataset



(b) Nottingham dataset



(c) J. S. B. Chorales dataset

Figure 5.20: Performance on consonance perception metric

All songs are taken from the test sets of the three datasets and the models used to fill the gaps in the songs have been trained on the training set of the same dataset that the song belongs to.

Figure 5.21 shows an extract from Prelude No. 3 in G Major, Op. 28 by Frédéric Chopin included in the test set of Classical Piano. This song features a lot of fast notes in succession which are difficult to predict. The models, in fact, do not manage to replicate the pattern of the melody, but they fill the gap with some long notes which are somehow coherent with the tonality of the song. This is certainly a better result than the one provided by the baseline model; however it is noticeable that the main issue in this case is that the models are not capable of replicating very fast notes.

Figure 5.22 shows a song taken from Nottingham test set, which is structured by a melody accompanied by a sequence of chords in regular cadence. From the piano roll it is visible that both neural network models manage to recreate a chord accompaniment which corresponds to the real chord; however they miss the chord change at beat 77 and they add some unnecessary notes from beat 74 onwards. Both models also try to recreate a melody for the very first beats, which is then interrupted; this could reflects the fact that the predictions on first timesteps can be more accurate than the ones on furthest timesteps.

Figure 5.23 shows a choral from Johann Sebastian Bach which is entirely made of 4 voices. In this case the feedforward model seems to return more complex predictions with many note changes with respect to the recurrent model which holds the same chord for many timesteps. Also this time the baseline model seems to be outperformed by the neural network ones.

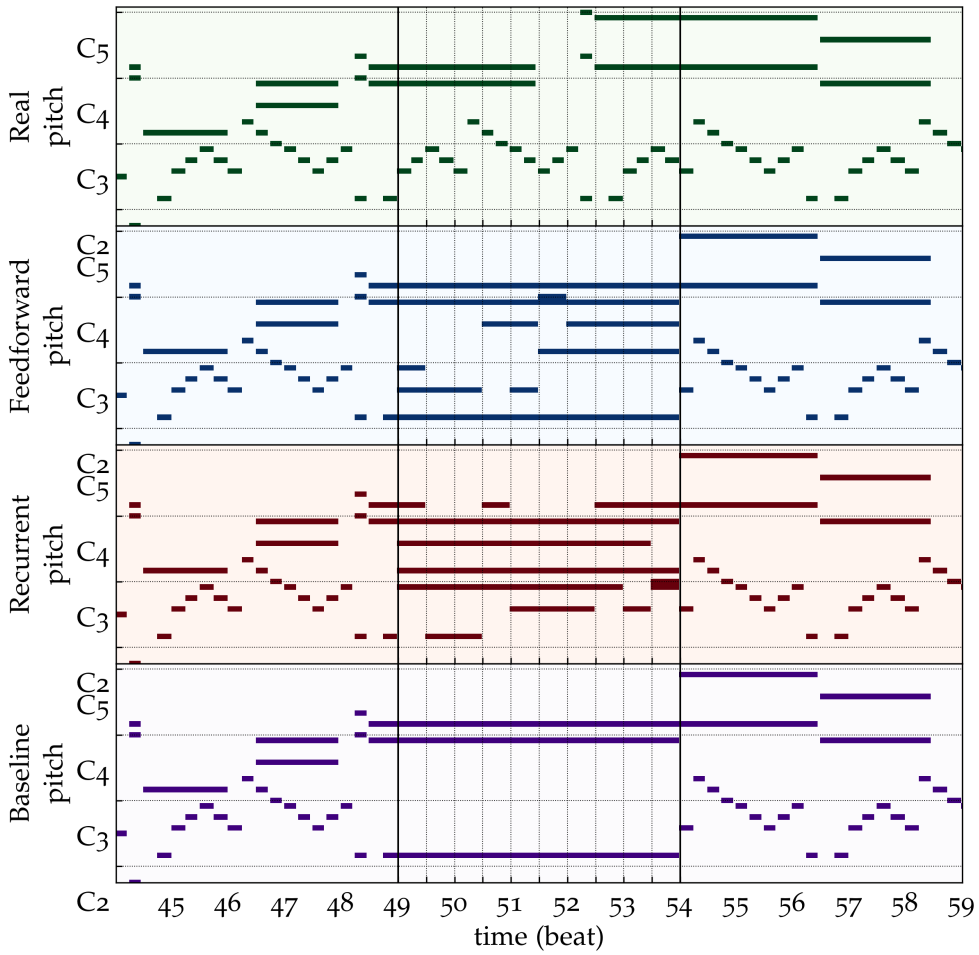


Figure 5.21: Prediction on Prelude No. 3 in G Major, Op. 28 by Frédéric Chopin from Classic Piano dataset

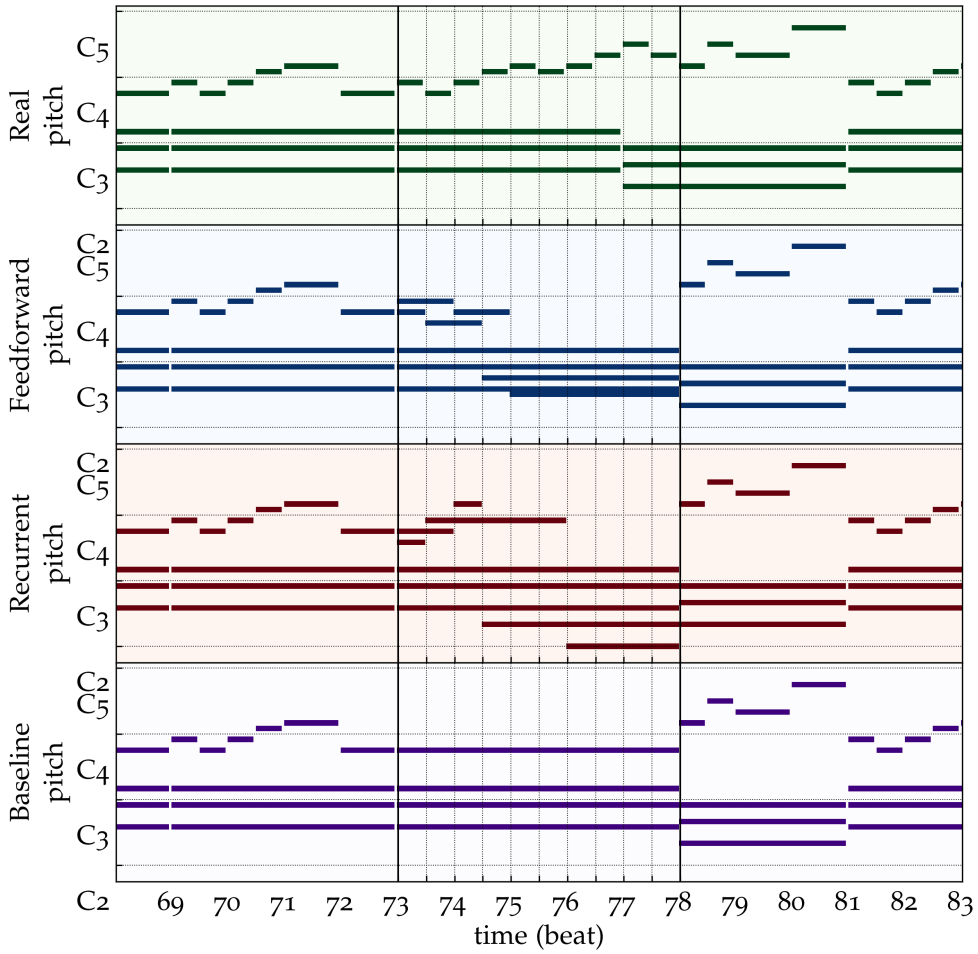


Figure 5.22: Prediction on a folk song from Nottingham dataset

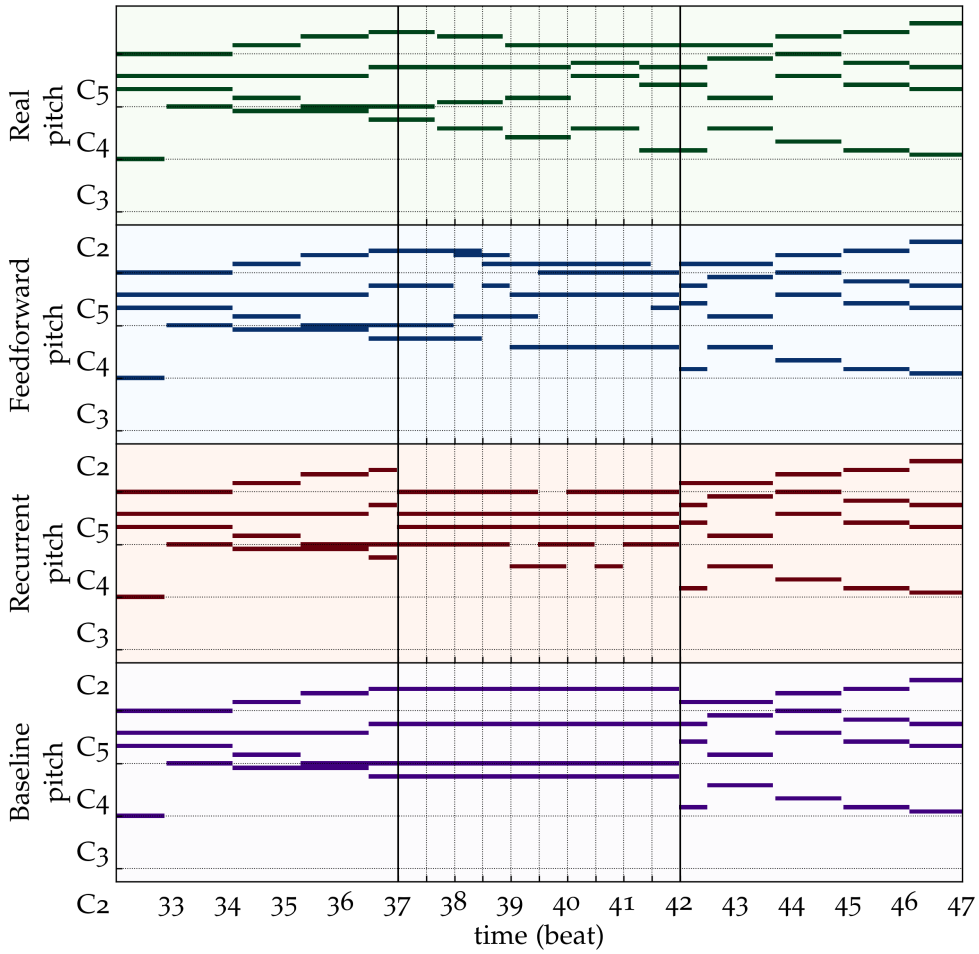


Figure 5.23: Prediction on J. S. B. Chorales dataset

6 CONCLUSION

In this thesis the concept of networked music performance has been introduced, underlining the major problems in dealing with an audio streaming with extremely strict requirements in term of latency and delay. Techniques as packet retransmission and error correction are not suitable due to the intrinsic nature of the audio streaming which has to be in real-time in order to allow musicians to play together in synch.

To tackle such issues, a method for filling possible gaps in the audio playback by using predictions generated by machine learning models is proposed. In particular, the models discussed in this work consist of feed-forward and recurrent neural network architectures, which take as input a window of timesteps from the audio playback and return as output the predictions for the next window of timesteps in which the gap may occur. In this way, predictions generated by the models can be used to patch the audio playback when it is needed.

To evaluate the performance of the models, they are compared with a baseline model which repeats the notes from the last available status of the music track before the gap. More specifically, different metrics are considered: the AUC-ROC which tells the capability of the models to discern between notes that are likely to be played and very unlikely to be played; a custom metric based on consonance perception which evaluates the consistency of the models from a musical point of view, favouring them if they are able to be 'in tune' with the real music and penalising them when their predictions are 'out of tune'.

It is demonstrated that both neural networks models are able to outperform the baseline one in almost every situation, providing predictions that are more complex and musically more consistent to the real track with respect to the notes generated by the simple baseline. Performance may vary depending on the dataset selected: the Classical Piano dataset is resulted to be the most difficult to model and predict, while Nottingham and Chorales by J. S. Bach are resulted easier due to their musical homogeneity and lower complexity of songs.

The improvement given by artificial neural network models in term of AUC-ROC is around 20% over the baseline for almost every future timestep, while the custom metric based on consonance perception shows that the proposed neural networks models perform better on the furthest timesteps. Also piano roll representations reflect this aspect, since predictions given by the machine learning models result to be visually more structured and complex, which, even if the original melody of the track cannot be correctly predicted, provide an alternative track from which it is possible to patch the errors and gaps in the playback of the audio streaming. In addition, the models are not able to replicate very fast notes or to continue sequences of

single notes like scales and arpeggios, but they are capable to extend chords and respect the tonality of the song.

Considering possible future developments, since the datasets used for training and testing the artificial neural networks influence the final performance of the models, it could be useful to include more data in the database, possibly incorporating more musical genres like pop, rock or blues to have a more accurate view of the general behaviour of such models. Moreover, since the framework proposed in this thesis has concerned only the part of prediction and music generation starting from MIDI tracks in the form of piano roll representation, the system by which those tracks are streamed and patched has to rely on an external software which has not been treated in this work. Such external software would be in charge to decide when to ask predictions from the model, e.g. at every timestep or after some interval, and decide which timesteps from the output window should be used to fill the gaps. For these reason, a next step would include integration of the model architecture with a streaming system by which it would be possible to perform more advanced and practical tests using a real-time approach.

ACRONYMS

AI Artificial Intelligence
Intelligence demonstrated by machines which imitate human behaviour.

AUC Area Under the Curve
Definite integral of a curve, representing the underlying area.

DL Deep Learning
Machine learning methods based on artificial neural networks.

FPR False Positive Rate
Ratio between the number of negative samples classified as positives and the number of negative samples.

GRU Gated Recurrent Unit
Gating mechanism in recurrent neural networks that controls which information from the past is preserved during training.

LSTM Long Short-Term Memory
Mechanism in recurrent neural networks which combine short-term memory blocks to create a long-term memory.

ML Machine Learning
Techniques and statistical methods which enable machines to learn from experience.

MIDI Musical Instrument Digital Interface
Connectivity standard for transferring digital instrument data.

NMP Networked Music Performance
Real-time interaction over a computer network that enables musicians in different locations to perform as if they were in the same room.

RNN Recurrent Neural Network
Type of artificial neural network that involves directed cycles in memory.

ROC Receiver Operating Characteristic Curve
Graphical plot which illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

TPR True Positive Rate
Ratio between the number of positive samples classified as positives and the number of positive samples.

UDP User Diagram Protocol
Connectionless communication protocol used for purposes where error checking, error correction and packet retransmission are not necessary.

BIBLIOGRAPHY

[1] Alexander Carôt, Pedro Rebelo, and Alain Renaud. "Networked Music Performance: State of the Art". In: *30th AES International Conference on Intelligent Audio Environments* (Saariselkä, FI, Mar. 15–17, 2007). New York, USA: Audio Engineering Society, Mar. 2007.

[2] Chris Brown and John Bischoff. *Idigenous to the Net: Early Network Music Bands in the San Francisco Bay Area*. 2002. URL: http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html (visited on 11/2020).

[3] Chris Chafe et al. "A simplified approach to high quality music and sound over IP". In: *COST G-6 Conference on Digital Audio Effects (DAFx-00)* (Verona, IT, Dec. 7–9, 2000). Dec. 2000, pp. 159–164.

[4] Alexander A. Sawchuk et al. "From remote media immersion to Distributed Immersive Performance". In: *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence* (Berkeley, California, USA). ETP '03. New York, USA: ACM Press, Nov. 2003, pp. 110–120. ISBN: 978-1-58113-775-0. DOI: 10.1145/982484.982506.

[5] *SoundJack: Real Time Online Music*. URL: https://ianhowellcountertenor.com/soundjack-real-time-online-music (visited on 12/2020).

[6] Chrisoula Alexandraki et al. "Towards the implementation of a generic platform for networked music performance: The DIAMOUSES approach". In: *International Computer Music Conference* (Belfast, Norhern Ireland, GB, Aug. 24–29, 2008). Michigan Publishing, Aug. 2008, pp. 251–258.

[7] Chrisoula Alexandraki and Demosthenes Akoumianakis. "Exploring new perspectives in network music performance: The DIAMOUSES framework". In: *Computer Music Journal* 34 (2 June 2010), pp. 66–83. ISSN: 0148-9267. DOI: 10.1162/comj.2010.34.2.66.

[8] Adrien Ycart and Emmanouil Benetos. "A study on LSTM networks for polyphonic music sequence modelling". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference* (National University of Singapore Research Institute, Suzhou, CN, Oct. 23–27, 2017). Ed. by Sally Jo Cunningham et al. California, USA: ISMIR, Oct. 2017, pp. 421–427. ISBN: 978-981-11-5179-8.

[9] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. "A Predictive Model for Music Based on Learned Interval Representations". In: *19th International Society for Music Information Retrieval Conference* (Cité International Universitaire de Paris, Paris, FR, Sept. 23–27, 2018). California, USA: ISMIR, Sept. 2018, pp. 26–33. arXiv: 1806.08686 [cs.SD].

[10] Jonas Langhabel et al. "Feature Discovery for Sequential Prediction of Monophonic Music". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference* (National University of Singapore Research Institute, Suzhou, CN, Oct. 23–27, 2017). Ed. by Sally Jo Cunningham et al. California, USA: ISMIR, Oct. 2017, pp. 649–656. ISBN: 978-981-11-5179-8.

[11] Inbal Shapira Lots and Lewi Stone. "Perception of musical consonance and dissonance: an outcome of neural synchronization". In: *J. R. Soc. Interface* 5 (11 June 2008). Ed. by Richard Cogdell, pp. 1429–1434. DOI: 10.1098/rsif.2008.0143.

[12] Hermann Ludwig Ferdinand von Helmholtz. *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. Trans. from the German by Alexander John Ellis. 3rd ed. London, GB: Longmans, Green, and Co., 1895. 576 pp.

[13] *MIDI 2.0 Specifications*. URL: https://midi.org/specifications-old/category/midi-2-0-specifications-v1-1 (visited on 12/2020).

[14] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: *Proceedings of the 29th International Conference on Machine Learning (ICML-12)* (University of Edinburgh, Scotland, GB, June 26–July 1, 2012). Ed. by John Langford and Joelle Pineau. ICML '12. Madison, Wisconsin, USA: Omnipress, June 2012, pp. 1881–1888. ISBN: 978-1-4503-1285-1. arXiv: 1207.4676 [cs.LG]. URL: https://www-etud.iro.umontreal.ca/~boulanni/icml2012 (visited on 12/2020).

[15] *Wikipedia: Artificial intelligence*. URL: https://en.wikipedia.org/wiki/Artificial_intelligence (visited on 12/2020).

[16] *Wikipedia: Machine learning*. URL: https://en.wikipedia.org/wiki/Machine_learning (visited on 12/2020).

[17] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: https://neuralnetworksanddeeplearning.com/ (visited on 12/2020).

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* (Fort Lauderdale, Florida, USA, Apr. 11–13, 2011). Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Proceedings of Machine Learning Research. JMLR, Apr. 2011, pp. 315–323.

[19] Kurtis Pykes. *The Vanishing/Exploding Gradient Problem in Deep Neural Networks*. May 2017. URL: https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11 (visited on 12/2020).

[20] *Keras Sequential class*. URL: https://keras.io/api/models/sequential/ (visited on 12/2020).