CS407 Group Project Final Report

Not quite my tempo:
The Humanisation of Music via
Sentiment and Semantics Analysis

Team: To Be Determined
Project Supervisor: Matthias Englert
Year of Study: 4

Department of Computer Science
The University of Warwick

2 May 2023

Abstract

Procedural music generation has been a topic of interest since the discovery of artificial intelligence and continues to grow in popularity as techniques develop. One, often overlooked, limitation of generated music is the lack of human emotion in playback. A majority of solutions for music generation are intended to compose melodies and harmonics in symbolic representation such as MIDI, resulting in the music sounding robotic when played back.

Our research focuses on the development of an automatic music humanisation system, which improves the quality of generated music with minimal user intervention, such that it sounds as if it is played by a human musician. For this purpose, we provide a number of tools to create data for this task and explore a number of different approaches for music generation. In doing so, we highlight the key challenges in this field, implement and evaluate various methods across different systems, and introduce the most important tasks for future work.

Keywords: deep learning, signal processing, natural language processing, music, procedural generation

---

Contents

Introduction

Music is an integral part of many cultures and allows for meaningful expression, conveying thought and emotion. Traditionally, most music is created through the use of instruments or the human voice, but the invention of the computer opened up a new area for music generation. Right from the birth of the field of artificial intelligence, many have been applying it to understand and generate music [Longuet-Higgins, 1976, Friberg and Sundberg, 1993]. Music generation has progressed to the state where it can fool professional musicians into thinking the generated music was created by a human [Oore et al., 2020]. Many thought that the creation of music would be the last creation bastion of humanity that is unrivaled by machines, but every year brings new improvements, bringing automatic music generation closer to that of human-level performance.

We present a new task in this field named *Music Humanisation*, to intentionally introduce specific features to an existing piece of music according to some instructions. These instructions vary according to the specific task; they could be a finite set of symbols representing tempo changes, an unbounded textual input (e.g., 'Play the first half of the piece jazzy'), or something else. Notably, the style, tempo, dynamics, notes, and other characteristics of a piece should change. This task is considerably more complex than music generation, as it requires understanding the intent of the instructions and manipulating the piece accordingly, whilst preserving the overall structure of the piece.

A need to solve this task arises from the nature of musical composition itself. Composers often write instructions to accompany their music; instructions, typically abstract and generic, specify how a section should be played. Capturing such instructions would expand the capability of current music generation tasks, allowing anyone to instantly hear a piece with certain instructions applied, without requiring a professional musician to play it.

Our contributions are as follows. We introduce the task of humanisation and discuss challenges and similar problems in different fields. We developed various unique tools to facilitate future research in this area, which can be used to create accurate datasets and analyse the structure of music. Finally, we present a variety of machine learning models for humanisation and discuss future work in this field.

1.1 Music Humanisation

Given some sheet music, computers can easily play music by creating sounds for every note at the required pitch for the exact correct duration. However, pieces created in such a fashion sound remarkably different from as if a human played the piece. Humans do not follow such strict rules when playing music, and we combine our previous musical experience with contextual information to play it slightly differently, by introducing specific dynamics and tempo. This contextual information could be the emotion of the song, the instructions in the piece, or even the title. This act of mapping synthetic music to 'humanised' music is the result of creativity and has been thought difficult to model through a formal structure. Thus far the algorithmic basis for this process is unknown, and there are no automatic systems that perform similar tasks.

We introduce the task of *Music Humanisation* which we define as follows: given some synthetic music as input, which may be an exact computer-generated version of a piece, and instructions, create a humanised version of the piece that satisfies several conditions. The structure of the resulting piece must be similar to the original, yet it must sound as if it was played by a professional human musician. The humanised version should exhibit some level of variation and expressiveness, such as subtle changes in tempo, dynamics, and timing, that are characteristic of human performance. Additionally, the resulting piece must follow the input instructions. These instructions could be a discrete set of objects, perhaps chosen from a finite list indicating the desired tempo, or could be in an unbounded representation like text.

An efficient solution for humanisation could be integrated into modern composing applications, allowing musicians to instantly hear an accurate rendition of their piece in the style they wish. The closest currently-available solution to this is MuseScore's[1] *Soundfont* feature, but this frequently produces poor-sounding renditions and doesn't allow instructions.

Similar problems have been tackled in other creative domains. In artwork, [Gatys et al., 2015] proposed a method to combine styles of arbitrary images, allowing any picture to be transformed into artwork in the style of a specific artist. Through this procedure, they automatically learn features that can be used to map styles between each other. Another similar approach has been developed in music, where the generation of music from scratch can be conditioned on the style of specific composers [Mao et al., 2018].

In machine learning, this task can be viewed as a machine translation task. Synthetic music and humanised music can be thought of as separate languages, so we can apply machine learning to automatically find the mapping from synthetic to humanised. Thus, advancements in this field can be leveraged to aid in solving music humanisation. However, this comes with many challenges.

Pieces of sheet music can be very long, sometimes containing thousands of notes and rests; we shall refer to notes and rests as symbols. Mapping two long sequences is a significant challenge; not only does it result in a large computational cost, but also proposes an algorithmic challenge to ensure long-term structures across the piece can be identified. There has been very little research in machine translation for handling long sequences. Natural languages can be chunked and each section can be independently translated with little loss to performance [Banar et al., 2020]. This approach is not possible in music as the structure at a certain position often weakly depends on the entire piece. Structures often repeat, or are similar across the piece, thus we must consider the full-length sequences to not lose any information.

Additionally, our research shows that this is an extremely data-limited scenario. As we explain later in Section 3, there are few datasets with pairs of synthetic and humanised performances. Previous research has shown that complex models with lots of data are required to accurately model language. Alternative methods of training models are necessary.

The performance of a machine translation based model to map between sequences can be measured using industry standards, such as cross entropy loss, and connectionist temporal classification loss. Cross entropy loss will effectively measure the similarity between the generated and desired output, and is popular in many autoregressive generation tasks. Connectionist temporal classification loss doesn't require explicit alignment between the generated and desired output, which is advantageous for music as the order of symbols can change whilst giving an almost identically sounding performance.

It is also important to evaluate the performance of the model with less rigorous methodologies; it is possible that a humanisation model will have a large cross entropy loss error, but will sound better than the target. However, this is difficult since the quality of music is known to be highly subjective. To mitigate the impact of this, we suggest a pseudo Turing Test as a method to evaluate the 'humanisation' aspect of a humanisation model [Turing, 1950]. A judge will be given several different human performances of a

[1] https://musescore.org/en

piece with a generated humanised version. They will either be asked to identify the generated humanised version, or in their opinion the best sounding piece.

1.2 Aims

Our primary objective was to research and develop preprocessing techniques and systems to aid future research in music humanisation. Since this is an unexplored area, we aimed to tackle fundamental problems in preparing datasets and developing tools for supporting future research in this area. Our systems were designed in such a way that aid users without musical backgrounds, such that they can be used by anyone.

The first of which pertains to instructions: we allow instructions in any format, interpreting them using natural language processing. These instructions must then be translated into the correct format used by our encoding, so they can be interpreted by our models.

Similarly, users lacking technical knowledge may not know which parts of a piece to change. As such, for each given input of synthetic music, we aimed to create a sectioning model that visualises the different sections in the piece and where they recur, if at all, later in the piece. Users can then specify which section they wish to change, rather than specifying time points.

Additionally, whilst there is a wealth of datasets for tasks generating music, the same cannot be said for music humanisation. Datasets require pairs of synthetic and humanised performances, which is difficult and laborious to collect. There are even fewer examples with contextual instructions for each piece. Thus, we aimed to develop two additional means of collecting data with labelled instructions. Using optical music recognition we can identify written instructions on the scores themselves. By reverse engineering an encoding of a performance, we can find instructions at specific sections by analysing how the piece is played. Whilst these require datasets in themselves, such data (images of sheet music for the former, and human performances for the latter) is easier to obtain than transcriptions of the instructions.

Our secondary objective was to develop some initial machine learning models for solving music humanisation to identify what can be achieved at present, and present potential areas for future research to explore. The majority of our research in this area was centered around researching and implementing computationally-efficient models, due to the inherently difficult nature of this task. We explored a variety of models to evaluate the performance of different paradigms of automatic music generation.

Due to the limited development time of the project, we will be restricting the scope and complexity of our research by only focusing on classical piano music. Classical pieces of music have distinct composition techniques and structure when compared to modern music, and thus need to be studied differently. Additionally, given instructions must be in English. We can summarise our aims with the following success criteria:

1. Research and evaluation of current systems in automatic music humanisation;
2. Development of a system to section a music piece;
3. Development of a system to extract instructions from images of scores;
4. Development of a system to extract instructions by reverse engineering them from pairs of scores and performances;
5. Development of a system to translate generic instructions into those used for music;
6. Development of music generation models to create performances given a score;
7. Evaluate and summarise what should be explored in the future.

1.2.1 Testing

To evaluate the success of our research, we devised several tests we would conduct throughout.

Following the humanisation methodology proposed in Section 1.1, we planned to use a loss function and responses from a study to evaluate our humanisation machine learning models. The study was planned to include the 'Turing Test' questions, as well as asking the participant questions related to the musical features of the piece, to learn more about the nuances of the performances produced by the system. We aimed to ask people from a variety of musical backgrounds to test if there is any correlation between answers and the volunteer's musical ability.

We also aimed to test our usability systems: instruction interpretation, sectioning, and our data generation systems: OMR system and reverse engineering. These are tested systematically, using white box and black box tests, without participants. The exact details of how systems are tested is specific to the system and decided by reviewing appropriate literature to use industry-standard approaches. These will be discussed in more detail in the respective section for each tool we developed.

Background

The task of music humanisation inherently has much in common with many other fields. In this section we provide background for the theory and technologies we used when researching and developing this project. We will first introduce a light background in music theory and how music is represented computationally. We then explore relevant fields in machine learning for our research, including optical music recognition and music generation. This second section is a result of a literature review we conducted in the first few weeks to survey the state of the field and understand employed techniques.

2.1 Music Concepts

This section details the necessary information required to understand music within the context of an automatic music humanisation system. A more in-depth explanation of music theory has been written by [Schmidt-Jones, 2013].

2.1.1 Music Notation

Music commonly exists as audio, represented in the form of sound waves, which can be perceived by human ears. However, audio is not a standardised representation of music as it cannot be easily manipulated by other musicians. A score containing music notation is a more general format to allow performers to create music, and convert it into audio that can be enjoyed by listeners in the general public.

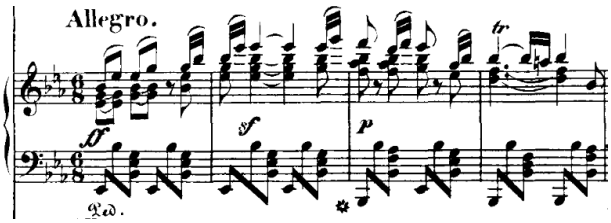


Figure 2.1: An example sheet music from the beginning section of Beethoven's Piano Concerto, Op.73, Movement 3

Sheet music is a collection of music notes and annotations, and it is a widely used format for delivering a human-readable, standardised music representation to musicians. An example of such is shown in Figure 2.1. It is typically paper-based containing music note, and annotations, indicating how to play the piece. Since the birth of the computer, we have seen a number of machine-readable music formats had be developed, allowing for easy replayability and manipulation of such data.

Music notes are housed by a set of five horizontal lines and four spaces called a staff. The horizontal position of each note defines the order they are played, whereas the vertical position defines the pitch they should be played on a musical instrument. In addition, the left-hand side of the staff defines time

and key signature. A time signature, denoted by a fractional number, specifies the number of beats in each measure, as well as the length of one beat as the basic time unit in music. A key signature, denoted by a collection of sharp ♯ or flat ♭ symbols, indicates the music scale the piece is intended to be played.

Annotations are instructions guiding how a musician should play each note. They help musicians to control three key components in music: time, tempo and dynamics. Time annotations indicate changes in the length of notes. Tempo refers to the speed of playing music, and is often expressed in beats per minute. Dynamics define how much force should be applied to an instrument. Annotations may also instruct performers on other auxiliary components; for instance, pedal control for piano music. An example of annotation can be seen in Figure 2.1, where the 'allegro' in the top left tells the musician to play at a brisk speed.

2.1.2 Classical Music

The structure and composition of classical music is substantially different from that of modern music; understanding these differences gives us insight into how we should approach humanisation for this genre of music.

Primarily, a complete classical music piece has a number of chapters, and each chapter is typically referred to as a movement. The composer then determines the structure of each movement, and further divides a movement into a number of parts consisting of different musical themes. There is a standardised practice, known as a musical form, for classical composers to decide their approaches to partition a movement and arrange each part.

Polyphony is one of the most widely used composing techniques in classical music. In a typical monophonic music setting, piano music has two staffs. There is one for each hand, where the right hand is on the high-pitch side of the keyboard and the left is on the low-pitch side. Traditionally, the high-pitch is responsible for the melodies, and the other for the accompaniments.

In a polyphonic setting, both staffs can contain melodies and accompaniments at the same time, and music notes are played simultaneously.

2.1.3 MIDI

MIDI is one of the most popular symbolic music formats, as compared to an audio file such as *mp3* or *wav*, which are not symbolic. A MIDI file preserves the exact notes, rests, and symbols within the piece. This allows for easy manipulation of the music for both composers and machine learning models. A visualisation of how MIDI encodes notes is displayed in Figure 2.2.

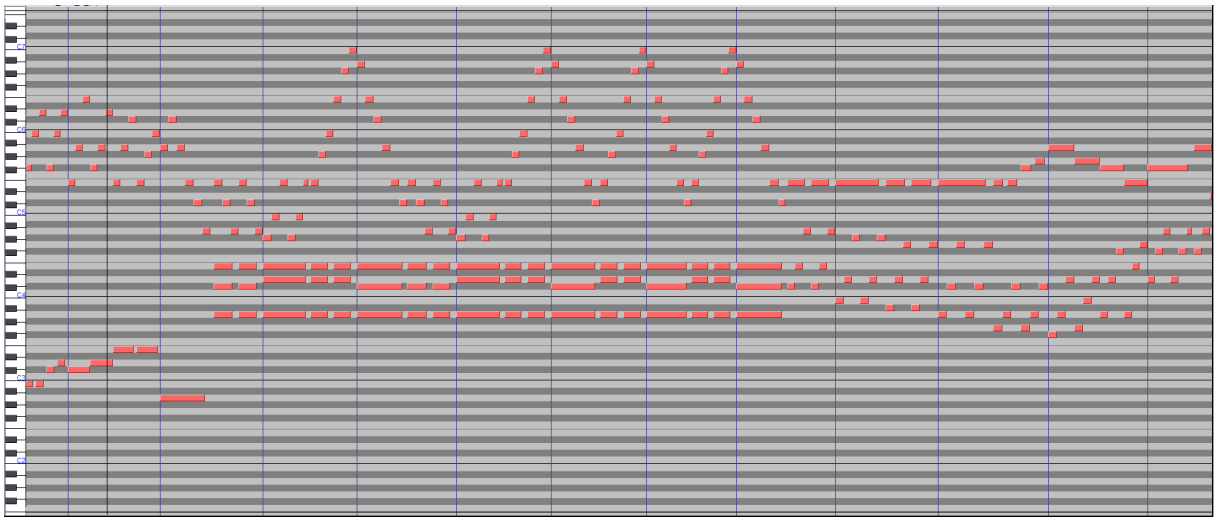


Figure 2.2: MIDI visualisation of music notes from a section of Schubert's piano sonata, D.959, Movement 4. Each horizontal orange bar represents a music note with varying length; x-axis denotes time and y-axis denotes pitch.

A MIDI file encodes each note with a number of properties. Each MIDI property value is commonly an 8-bit unsigned integer. The pitch of a music note is identified by a unique MIDI note number; for example, the centre note A4 with a frequency of 440Hz has a MIDI note number of 69. Dynamics in MIDI are typically referred to as velocity, which ranges from 0 to 127, with 127 being the loudest.

The tempo in MIDI is specified in pulses per quarter note (PPQ), rather than beats per minute (BPM). The main reason for this is that the definition of a beat depends on the time signature. For example, we can have a quarter note per beat or an eighth note per beat. This can be inconvenient whenever the time signature is changed during the playback of the music piece. Therefore, a unified beat of a quarter note is used in the MIDI timing system. A universal formula can be used to convert between BPM and PPQ, as well as for calculating the absolute timestamp within a MIDI file. As the name suggests, PPQ defines a minimum division of time measure called 'pulse' or 'tick' in MIDI files, specified in milliseconds.

As a digital music notation file, a MIDI file specifies instructions to the computer for playing each note, but does not support direct playback as the format does not specify any sound data internally. To do so, an external sound synthesiser has to be used, that converts music notes in this format into audible sound waves.

2.1.4 MusicXML

MIDI is designed for low-level hardware communication between digital musical instruments. Consequently, any information that is not recognisable by a playback system is removed, including music instructions. MusicXML is an alternative to MIDI, which is designed to deliver more information than MIDI, for use on a desktop computer rather than embedded devices.

As its name suggests, MusicXML is based on XML file format, and all properties are specified via a number of XML tags. An attribute tag defines a number of properties for a piece of music. This includes key and time signature, and the arrangement of staves.

```
<attributes>
  <divisions>12</divisions>
  <key>
    <fifths>3</fifths>
    <mode>major</mode>
    </key>
  <time symbol="common">
    <beats>4</beats>
    <beat-type>4</beat-type>
    </time>
  <staves>2</staves>
  <clef number="1">
    <sign>G</sign>
    <line>2</line>
    </clef>
  <clef number="2">
    <sign>F</sign>
    <line>4</line>
    </clef>
</attributes>
```

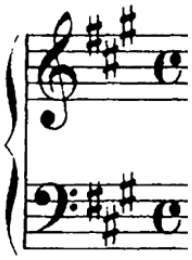


Figure 2.3: An attribute in MusicXML that specifies a piece with a G and F clef, and time signature of $\frac{4}{4}$ in A major, which is equivalent to the sheet music on the right.

Notes can then be added to each measure in a staff, via the *note* tag. Similar to MIDI, MusicXML allows defining the pitch and length of a note. Additionally, supplementary information can be specified to stylise the display of a note, including the beam direction, as well as note decoration, such as 'slur'.

```
<note default-x="110.51" default-y="-15.00">
  <pitch>
    <step>C</step>
    <alter>1</alter>
    <octave>5</octave>
    </pitch>
  <duration>9</duration>
  <voice>1</voice>
  <type>eighth</type>
  <dot/>
  <stem>up</stem>
  <staff>1</staff>
  <beam number="1">begin</beam>
  <notations>
    <slur type="start" placement="above" number="1"/>
    </notations>
</note>
```

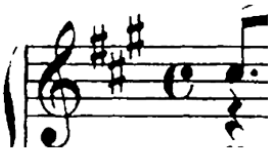


Figure 2.4: A *C#4* dotted eighth note specified by MusicXML.

The main feature that distinguishes MusicXML from MIDI is the ability to add information to the music, beyond the notes. This extra information can be used as music instructions to guide musicians to play the piece in a certain way and provides additional data for music-related research.

```
<direction placement="above">
  <direction-type>
    <words default-y="41.00" font-weight="bold" font-style="italic" font-family="Times New Roman" font-size="12">Allegro moderato</words>
    </direction-type>
  <staff>1</staff>
  </direction>
```

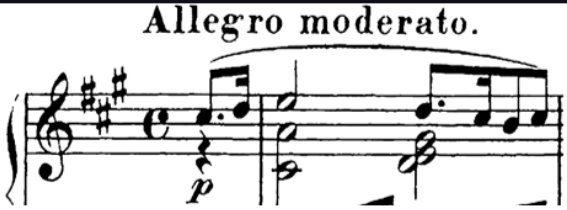


Figure 2.5: An instruction *Allegro Moderato* is placed above the G staff.

2.2 Music Generation Models

The following sections detail research done on music generation models, and their potential suitability to our task. Music generation is a field similar to humanisation, as both are concerned with the automatic generation of music.

2.2.1 MuseGAN

MuseGAN [Dong et al., 2018] is a music generation model, that uses a Convolutional Neural Network (CNN) based Generative Adversarial Network (GAN). This model has been shown to effectively generate long-form polyphonic music with multiple instruments. In particular, we were interested in this model primarily due to its ability to generate music from existing music pieces given by the user. This relates closely to our task of humanising an existing piece. The GAN architecture is the prototypical model for generative tasks [Goodfellow et al., 2020]. On the other hand, whilst most models use a type of Recurrent Neural Network (RNN), this model uses CNNs instead, which have their own unique benefits in this setting.

Convolutional Neural Networks

CNNs work principally using the convolution operation. Put simply, they apply a filter to some given input, extracting a feature map. This feature map contains information that extracts only the relevant features, allowing the model to better learn the task. As such, these are widely used in image classification, in which the meaning of these filters is quite intuitive; the filter is applied across different subsets of the pixels, thus transforming it into a new image.

Over time, the model is trained to learn the correct filters to extract the relevant features. Training to learn these filters is significantly more efficient than learning the weights of individual neurons as in a typical neural network. Consider an image, with size $(128, 128)$. If we want to capture information from each pixel, we would need $128 \times 128 = 16384$ neurons at the input layer. This means training just as many weights as well. In comparison, if we use a $(5, 5)$ filter, we are training only $5 \times 5 = 25$ weights; each of the values in the filter. This is not only more efficient and thus requires a smaller size dataset, but can also prevent overfitting.

Due to the nature of a CNN, they're able to work well on high-dimensional data and longer sequences. This usually causes problems for RNNs; as the sequence becomes longer and more complex, retaining information from earlier parts becomes more difficult. A CNN is less susceptible to this as, intuitively, it looks at the sequence as a whole.

Generative Adversarial Networks

GANs [Goodfellow et al., 2020] consist of two components: a generator and a discriminator. The generator learns to create new data synthetically, while the discriminator learns to distinguish between synthetically created examples and real examples. Both networks are trained together in a zero-sum game; the aim of the generator is to fool the discriminator, whilst the discriminator aims to discriminate between the examples created by the generator and real-world examples. A large loss for one network necessarily corresponds to a low loss for the other network; if the generator can fool the discriminator it will have a low loss, as it achieved its task, whilst the discriminator will have a larger loss, as it failed its task, and vice versa. As these two networks converge, the generator should be able to fool the discriminator 50% of the time; this corresponds to the discriminator not being able to do better than simply randomly guessing.

GANs solve an unsupervised learning task: given real-world examples, generate realistic looking examples. As such, we don't require labels for these examples; the generator implicitly tries to learn

the underlying distribution of the data D, so that it can randomly sample from it when asked for a new example.

Variable-sized input

This previous model cannot be directly applied to our task, for a few reasons. The first is variable input size. We aim for the final system to work with pieces of any length, but CNNs require a fixed-size input.

Indeed, most models, with the exception of RNNs, are not able to inherently deal with variably sized inputs. A neural network consists of layers, each with a number of neurons - typically a fixed number of them. Whilst a layer implementing convolution does not need to be of a fixed size, CNNs typically contain at least one fully connected layer, at the end. A fully connected layer is one where each input contributes some weight to each output. When used for image classification, this is typically the last layer, to converge to some fixed number of classes to classify the image as. In such a fully connected layer, the number of input neurons and output neurons is fixed. This meant, to use this model, separate modifications would be necessary for the generator and discriminator.

Pooling is used to reduce the dimensionality of some data, by sampling subsets of it. There are many kinds of pooling, but most fall into either average or max pooling. Consider an image of size $(4, 4)$ which we want to reduce to a size of $(2, 2)$. Average pooling can be thought of as averaging over each corner, whilst max pooling can be thought of as taking the maximum of each corner. This is better illustrated in Figure 2.6. We can use pooling to solve the problem of variable-input size. However, we cannot use generic pooling. Similarly to regular convolutional layers, we cannot fix an output size unless we know the input size. There is a special kind of pooling to deal with this, called adaptive pooling; a generalisation of Spatial Pyramid Pooling (SPP) [He et al., 2014]. Adaptive pooling allows us to specify an output size, without necessarily knowing the input size. This allows CNNs to overcome their inability to handle variably sized inputs, and thus be applicable to this task.

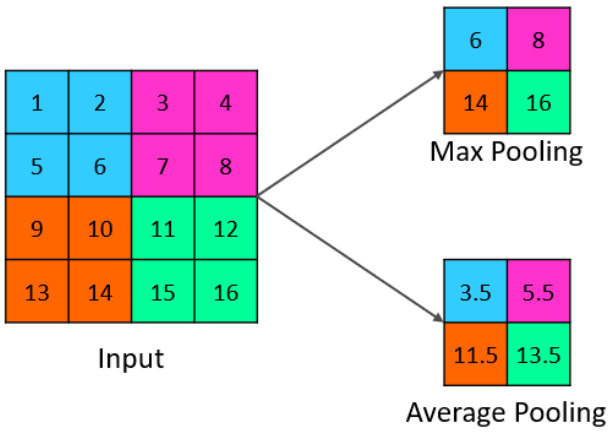


Figure 2.6: An example of max and average pooling on a $(4, 4)$ input image; colours illustrate the location where the sampling kernel is applied.

2.2.2 Transformers

The transformer is a recently-proposed architecture that has excelled in a wide domain of machine learning tasks [Vaswani et al., 2017, Lin et al., 2022]. It has two sections, an encoder and decoder, which make it natural for tasks mapping from one sequence to another, such as machine translation. These sections can also independently act as models for specific tasks. The decoder alone is fantastic for autoregressive tasks, making it key for much of modern natural language processing [Brown et al., 2020].

The encoder and decoder are fundamentally composed of their own repeating 'blocks' that can efficiently be stacked to increase the complexity of the model.

Attention is a key component of every block, allowing the model to attend to any value from the previous layer. This is important for manipulating data with long and complex structures, such as language. However, since it has to attend every timestep to every previous timestep, this gives time complexity $\mathcal{O}(n^2)$ with respect to the input sequence length for every use of attention. As seen with recent progress in language modelling, the complexity of the attention mechanism is often the bottleneck for increasing the complexity of the model [Black et al., 2022]. Additionally, whilst the scalability allows for huge models to be constructed, their size makes them hard to optimise. The residual connections and layer normalization present in the vanilla architecture somewhat alleviate this problem [Vaswani et al., 2017].

Many variants of transformers have been proposed to increase their computational efficiency on long inputs. These can be broadly split into two types: local focusing models, and global approximation models. The first apply attention to sections of the input, and ignore the rest. Whilst this gives a significant improvement in memory usage, automatic methods of identifying which areas are best to attend are poor and fail to be robust across distributional shift, i.e., when the distribution of data changes as it is deployed into the real world [Beltagy et al., 2020, Dai et al., 2019]. The second type is when the model attempts to weakly attend to the whole input, either through compressing the input or using sub-polynomial attention mechanisms. These have been shown to struggle to model correlations and complex structures in text compared to the original attention mechanism [Katharopoulos et al., 2020].

Despite their successes, none of those approaches focus on overheads from memory access (IO), causing their wall-clock speedup to be smaller than intended. Recently, [Dao et al., 2022] proposed the Flash Attention mechanism, allowing models to be trained significantly faster with a smaller memory footprint by computing attention with far fewer memory accesses. It computes exact attention without approximation and has time complexity linear with respect to the sequence length.

Transformers are naturally suitable for music generation due to their repeated attention capturing complex structures, and have been shown to be able to produce long sequences of coherent music [Huang et al., 2018]. However, due to their large memory and computational footprint, they often are overlooked. Recent proposals have been shown to significantly decrease this footprint yet most approaches still have polynomial time complexity [Zeng et al., 2021, Chou et al., 2021].

2.2.3 Museformer

A recently-proposed music generation model is Museformer [Yu et al., 2022], a decoder-only transformer model that improves upon existing transformer models with a novel attention scheme designed to focus the model on structurally significant information. At the time of writing this is the state-of-the-art transformer-based model for symbolic music generation, with the added benefit of efficient memory usage for the very long token sequences that are required to represent a full piece of music.

Attention Scheme

The key design idea of Museformer is the 'fine and coarse grained' attention scheme. The main idea of this is that a token is not directly attended by all other tokens. Instead, a token is only directly attended by tokens of bars that are structurally related to the current bar (e.g., the tokens of the previous bar) and then is also attended by a summarisation token for all other bars in the sequence. This greatly reduces the complexity of the attention scheme from that of full attention, while reportedly improving the models ability to generate structurally consistent music.

This attention scheme relies on two main ideas: the summarisation step and structurally related bars. The former relies on the tokenisation method inserting a marker token at the end of a bar, which

then attends to all the tokens within the bar it 'summarises'. The model can then look at that token instead of all the previous bar tokens to get some general information without having to look at the entire sequence if it is not deemed structurally important. The next concept is less concrete, the paper describes structurally related bars as bars that are 'likely to be repeated by the current bar to be generated'. There is no certain method of finding bars that will be repeating exactly, but the paper proposes a simple Jaccard similarity metric $l_{i,j}$ between pairs of bars, which were then averaged across the entire training set to find a general distribution of bar similarities.

$$l_{i,j} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

The identified common structure-related bars (previous 1st, 2nd, 4th, 8th, 12th, 16th, 24th, and 32nd bars) will likely not cater to every song, but as a general structure allows the model to generate more regularly structured music in line with its training set.

We cannot directly implement this model for our task due to the need for an encoder to help with the translation mapping between score and performance, as these may be substantially different in the training dataset. However, we will likely be able to make use of this attention scheme at least in the decoder block of our architecture to hopefully improve our generated piece's structure.

2.3 Optical Music Recognition

Optical Music Recognition (OMR) enables the automatic recognition and digitization of musical scores. Applying OMR allows us to transcribe scanned images of sheet music into machine-readable musical formats, such as MIDI or MusicXML. This is a challenging task as it requires solving several problems: identifying all symbols within a score, finding the relative positions of all symbols, and creating an encoding of such in the desired format. OMR has several different variants, depending on what symbols are being identified. Some studies just detect notes and rests, whilst others additionally detect barlines [Na et al., 2015].

Historically, OMR approaches have followed a typical framework of four stages: image preprocessing, recognition of musical symbols, reconstruction of required music information, and construction of the required musical format [Rebelo et al., 2012]. Extensive preprocessing has often been conducted due to the computational complexity of transcribing an entire piece, and the inherent noise associated with the task. Much work has been done to reduce the effect of barlines, staff lines, staccatos, and irrelevant text from damaging downstream performance [Shatri and Fazekas, 2020]. After noise reduction, symbols are then isolated from one another, classified, and then paired to construct the resulting music piece and encoding. Before deep learning dominated this field, support vector machines were often used for classification due to their ability to efficiently handle high-dimensional data.

Recently, deep-learning-based approaches have given the best performance in all required stages of OMR [Edirisooriya et al., 2021]. Some have viewed OMR as a machine-translation task, allowing them to use encoder-decoder-based architectures which have often given impressive results [van der Wel and Ullrich, 2017]. This allows for an end-to-end architecture, which has the model learn to perform all required stages. Despite the recent success of transformers in many other domains, they frequently perform worse than LSTMs [Ríos-Vila et al., 2022]. The generic transformer model features an attention mechanism which allows it to attend to the entire input for every layer, however, this feature is quite redundant for OMR. Recognising a symbol only requires local, not global, information which bidirectional-LSTMs achieve with a simpler architecture.

2.4 Natural Language Processing

Natural language processing (NLP) is the study of techniques to allow computers to understand human language. There are a wide variety of applications of NLP, one such application being information extraction; to look for useful data within some text and format the data into a machine-readable format. Studying NLP is essential to our task, as we require the given instructions in harmonisation to be understood. Furthermore, as we can view music as a language, we can use insights in the field of NLP to improve our research.

The NLP pipeline defines a sequence of components to be used to achieve a specific task. Each of the components uses different techniques to solve a particular problem. In this section, we'll discuss components relevant to our research.

2.4.1 Tokenisation

Most machine learning architectures cannot parse raw textual input texts directly; they require such texts to be represented in some numeric form. Tokenisation is the process of converting a given text into a sequence of tokens, which are any sequence of characters or symbols that represent a meaningful unit in a text.

A typical example is word-based tokenisation, where each word is a token. For example, 'The quick brown fox' would be mapped to 'The', 'quick', 'brown', 'fox'. We could then represent this computationally using *one-hot encoding*. With a vocabulary V containing every word in the English language we wish to represent, we can create a unique vector for every word. For every word $w_i \in V$, we construct a vector $x_i \in \mathbb{R}^{1\times|V|}$. Each vector h_i contains a '1' in the ith position in the vector, all other elements are '0'. Below is an example of this encoding method:

$$\begin{aligned} \text{dog} &\to [1,0,0] \\ \text{cat} &\to [0,1,0] \\ \text{parrot} &\to [0,0,1] \end{aligned}$$

We could extend this simple method to tokenise music by taking our vocabulary as all the possible symbols in a score of music, which would contain every symbol for every possible pitch for every possible duration. Given a score of music, we could encode it into a sequence of vectors using one-hot encoding over our vocabulary. However, creating a unique vector for every possible symbol would result in a very large vocabulary. Instead, since each note contains the pitch, velocity, and duration, we could choose to independently tokenise each of such characteristics separately and then concatenate the final result. This would allow future models to learn more generalised representations for each token as they are shared between different notes.

The vectors given by one-hot encoding are basic and don't include any information about the represented symbol. All vectors are orthogonal to one another, thus there is no similarity between any of them. Ideally, we'd want to encapsulate some measure of similarity in our vectors, such that similar symbols will have similar vectors.

2.4.2 Word Vectorisation

Recently, many methods have been proposed to computationally represent words whilst inherently capturing their meaning and relationship to other words through their value.

Each word can be represented by a dense vector with the aim for similar words to be close to one another in Euclidean space. Furthermore, the vector between different words should represent their relationship. An example of these relationships is visualised in 2.7; we can see that the vector from the embedding of 'strong' to 'stronger' is similar to that from 'slow' to 'slower'. Word embeddings offer an

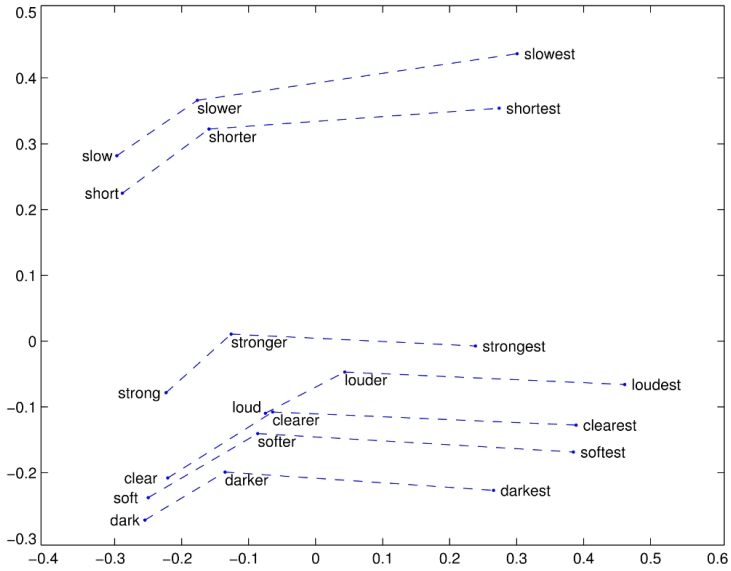


Figure 2.7: GloVe embeddings visualised, with word relationships highlighted

advantage over one-hot encoding in machine learning; they allow for the meaning of words to be inputted into a network, rather than the network having to learn such meanings itself.

There exist two main methods for creating these embeddings. A dataset containing text is required, which is known as a corpus. Unsupervised methods can learn such embeddings through a large corpus when analysing co-occurrence statistics, or through predicting neighbouring words in text [Pennington et al., 2014]. Alternatively, many modern architectures incorporate the task of learning word embeddings in their solution as an auxiliary task, which can result in high-quality embeddings [Radford et al., 2019, Brown et al., 2020].

However, the downstream success of using these embeddings largely depends on how we choose to tokenise text. If we choose to tokenise on a word level, then the token for 'brown' will be the same for both 'Dr. Brown' and the respective token in 'The brown cat'. Words in the English language often have more than one meaning, which presents a challenge for applying tokenisation. We are likely to face this problem when parsing instructions, as many piano annotations are in Italian. To prevent this problem we can apply **entity linking**; the process of disambiguation of words that can potentially come with different meanings when presented in different contexts.

2.4.3 Lemmatisation

Training an NLP model often involves the use of a large corpus. However, if the model attempts to search for some exact word in the corpus, there is a high chance that no such matching is found. This is due to different variations of word form used in real-life, while the corpus only defines words in their base form to save memory.

Informally, word forms are grammatical changes made to a vocabulary. Examples like plurals add suffix *s* or *es* to the end of the word, or regular simple past tense adds suffix *ed*. Lemmatisation is a process of reducing special word form back to its base form, such that it can be looked up from the corpus.

The simple approach of removing prefixes and suffixes from a word does not work practically. Words like *testing* may either be a singular noun itself, or the present participle of the base form *test*. Some word forms are irregular (like based form *write* whose past tense is *wrote* rather than *writed*) that should be handled differently.

2.4.4 Named Entity Recognition

Named Entity Recognition (NER) is the step of identifying entities in a sentence, such as name of a person and places. This is a key step for information extraction that allows the model to filter out unrelated information in a sentence.

Data

In recent years, many fields of machine learning have seen their datasets explode in size due to the proliferation of data generation and collection technologies. Music is a field that has also witnessed a significant increase in its datasets. The widespread availability of digital music platforms has resulted in a wealth of music data that can be scraped and processed for machine learning tasks. MuseScore[1] is a website that allows users to upload their own composed music which can be downloaded by other users for free. In 2017, there were 1.6 million songs that could be scraped[2]. Even older is the Million Song Dataset[3], which was released in 2011 and contains a broad range of different genres.

Unfortunately, there is very little data for music humanisation. We require examples of synthetic pieces paired with humanised performance pieces and instructions. Datasets containing such examples are laborious to make, as recordings of thousands of pieces from a professional musician are required, which must also be converted into MIDI or another amenable format. Alternatively, one could manually match recordings of music to the corresponding synthetic piece and ensure they're in the correct format. From all available datasets, we estimate there are less than 2000 examples of this data.

In this section we introduce the different tokenisation schemes we applied for representing our music, as each comes with its own advantages and disadvantages. We also discuss the different datasets that we used in our research and analyse their structure.

3.1 Tokenisation

The simplest tokenisation method for MIDI files is to simply represent each MIDI message as a token, i.e., such that all *note on*, *note off*, *velocity*, and *time shift* messages have a respective token. However, this often results in poor performance [Oore et al., 2020]. The duration of each note is not explicit and must be inferred from the time shift tokens between a *note on* and *note off*. Not knowing the duration of a note as soon as it has been played leads to worse music generation for models using this tokenisation, as the model cannot immediately distinguish between short and long notes. The *note on* and *note off* may be far away in the input; models will struggle to recognise the pair.

In response to this, [Huang and Yang, 2020] proposed two new tokenisation strategies. The first, TimeShift Duration, used *duration* tokens to represent note durations. They also proposed REMI, where they use *bar* tokens to explicitly indicate when a new bar is beginning, and *pitch* tokens to show the position of notes. Each note is represented using a secession of *pitch*, *velocity*, and *duration* tokens. REMI has been shown to lead to quicker convergence, and the explicit representation of *bar* tokens makes it easier to merge our instructions with the piece tokenisation. One major downside of REMI is that it results in extremely long sequence lengths. Musical pieces sometimes contain thousands of notes and rests, and REMI typically gives three tokens for each.

[1] https://musescore.org/en
[2] https://github.com/Xmader/musescore-dataset
[3] http://millionsongdataset.com/

The Octuple tokenisation strategy was introduced by [Zeng et al., 2021] to address the concerns of long sequence lengths. Each symbol is encoded using eight separate tokens, which fully encapsulate all information about such symbol. These tokens each have their own embedding procedure, and are then concatenated so each note is represented by a single embedding. Despite significantly reducing sequence length, the authors found no diminish in performance when used in music generation. Additionally, having each note contain full information, such as bar information, allows for much simpler manipulation of this data.

3.2 Instructions

To obtain instructions, we wrote a parser in Python to extract annotations from the sheet music[1], based on a previous implementation[2]. Sheet music often has accompanying annotations specifying how the music should be played in specific sections. Using the exact position of these instructions, we can then tokenise them along with the rest of the piece, and insert them in the correct position.

Ideally, we would extract the exact annotation and position within the piece. Since MusicXML doesn't explicitly store the position, we could calculate it by first finding which bar it's in, then summing the duration of non-overlapping notes until the instruction. However, due to the MusicXML format, this wasn't always possible. In music, every bar's time in every stave, apart from the first, must sum up to the time signature. However, MusicXML is sometimes incorrect with what stave a note is in, in some cases storing extremely low or high notes in a different stave in its internal structure. This results in our summing solution being misaligned. For example, in Figure 3.1, whilst all notes in bold should be played by the right hand and thus encoded in the top stave, the lowest notes are stored in the bottom stave.

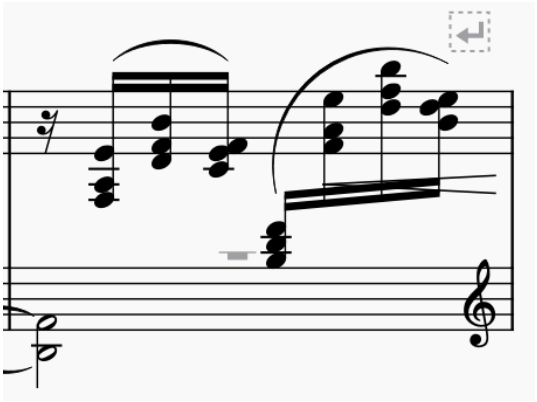


Figure 3.1: Visualisation of MusicXML example with incorrect formatting

We could remedy this by using MusicXML to extract the instruction, and then use the respective MIDI piece to find the exact location. However, finding the bar of an instruction is sufficiently accurate, and much less time-consuming. With the bar numbers, we merged the instructions with our MIDI tokens by inserting the instructions at the start of their respective bars.

Unfortunately, as we noted earlier, this data is only available in MusicXML files, not MIDI files. To address this we created three tools to extract instructions from a variety of formats, which we discuss in section 4.

[1] https://github.com/ojaffe/MusicXMLAnnotations
[2] https://github.com/sachindae/polyphonic-omr

3.3 Datasets

Modelling music is complex and requires large machine learning models to generate correct music according to the rules of music theory. Recently, the only models able to accurately model music are large recurrent neural networks or transformer-based auto-regressive models, each containing millions of parameters. As such, huge datasets are required to ensure they can learn the required generalized behaviour. Our task is more complex than just music generation, requiring more complex models and perhaps larger datasets.

To alleviate our data shortage, we use music generation to pretrain some of our models. By harnessing music generation for pretraining, we capitalize on the wealth of data available to establish a solid foundation for our models. These base models will then be fine-tuned on smaller humanization datasets specific to our task, effectively addressing our data shortage. Since our desired humanisation model requires an understanding of generating music to accomplish its task, pretraining it on music generation will allow it to learn a portion of its task.

This approach of transfer learning has been frequently used in other subject areas, such as computer vision. For example, medical image classifiers make use of pretrained deep neural networks and are refined for more specific tasks such as Alzheimer's stage detection and knee osteoarthritis assessments, along with a variety of other use cases [Zhuang et al., 2020]. Additionally, pretraining a generation model has also successfully been attempted in sign-language translation, due to the lack of data in the field [De Coster et al., 2021].

To reduce the complexity of our project, we aimed to follow a similar procedure, though we noted it may require a substantial time investment to engineer our model to allow for this process. Another time investment is that to achieve the benefits of pretraining our model for the generation task we need to be confident in its abilities to learn this task, which itself may require a long training process.

3.3.1 ASAP Dataset

In our research, we used the Aligned Scores and Performances (ASAP) dataset as our primary source of data for training models on humanisation [Foscarin et al., 2020]. This dataset consists of Western classical piano music, containing 236 digital musical scores paired with 1067 humanised performances. The scores are provided as MIDI and MusicXML files, and the performances are available as MIDI files and partially as audio recordings. It is noteworthy that the scores and performances are aligned with downbeat, beat, time signature, and key signature annotations, providing a comprehensive representation of the musical content.

We analysed properties of the dataset to evaluate suitability for our task. Note, we omit analysis of the synthetic and humanised examples independently, as the results were almost identical. The humanised performances had slightly longer sequences on average, but their overall length distributions were very similar.

We first looked at the average sequence length of different tokenisation schemes. In Figure 3.2 and 3.3 we can see histograms of sequence lengths from REMI and octuple tokenisation respectively. We see that REMI often gives extremely long sequence lengths of over 20000 tokens and has outliers up to 85000 tokens. Whilst REMI gives an average sequence length of 15000, octuple gives an average sequence length of 3422. This clearly motivated us to use octuple in our future models, as otherwise we'd have to prune the majority of the input.

After some manual data inspection, we realised that a significant number of examples had a large difference between their two sequence lengths, with the performance sequence sometimes reaching four times the length of the score. This was a result of the performance playing at a different tempo, which caused the resulting tokenisation to have different sequence lengths, even if the number of notes played was exactly the same.

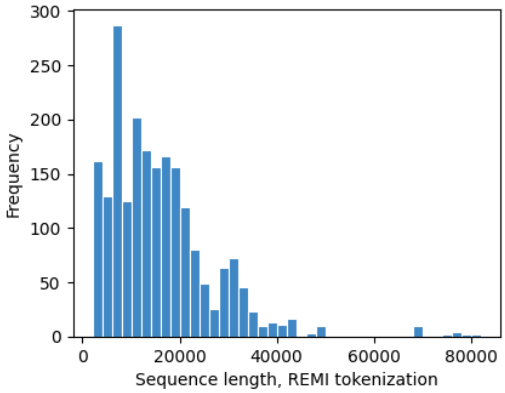


Figure 3.2: Sequence lengths of REMI tokenisation applied to the ASAP dataset

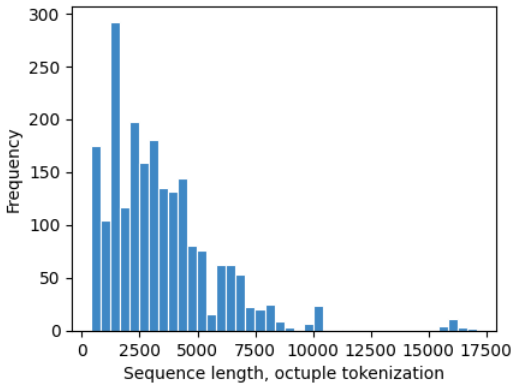


Figure 3.3: Sequence lengths of octuple tokenisation applied to the ASAP dataset

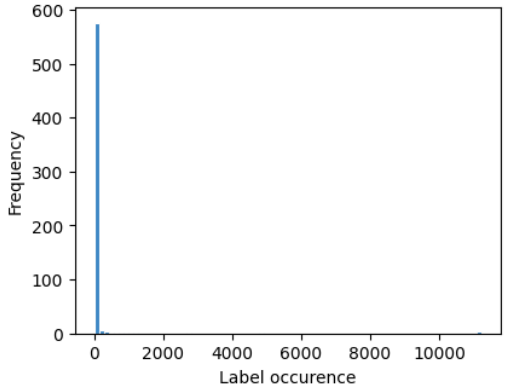


Figure 3.4: Histogram of instruction frequency

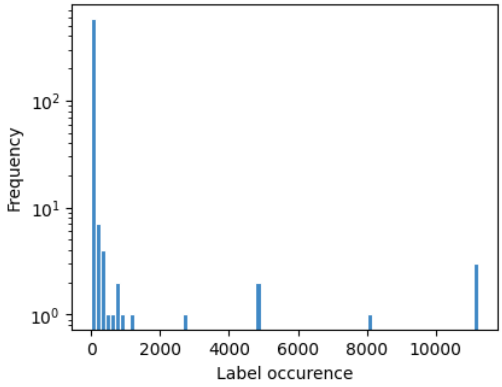


Figure 3.5: Histogram of instruction frequency, with log applied to frequencies

To see the variety of instructions, we passed all examples through our instruction parser and counted the occurrences of all present instructions, then plotted histograms to view their frequencies. In Figure 3.5 we see the original histogram, which shows us that most instructions occur extremely infrequently. This is likely because our instruction parser extracts most musical annotations which sometimes gives irrelevant information, like the musician's name. In Figure 3.5 we plot the same histogram and log the frequencies. We see that there are few instructions that commonly occur.

Figure 3.6 displays the most frequent instructions. We can see that they all relate to dynamics, unfortunately with none for tempo, which will cause modelling tempo to be difficult. To ensure future models have enough data to learn the required relationships, we will prune these instructions to only include those seen more than 1000 times. Future research could look into combining several infrequent instructions into one; we found that many parsed instructions had the same meaning, but were parsed differently giving different texts.

Instruction	Frequency
sf	11228
other-dynamics	11189
p	11148
f	8090
pp	4830
ff	4778
fz	2688
cresc.	1231

Figure 3.6: Most frequent instructions in ASAP dataset

3.3.2 ACPAS

The ACPAS dataset[1] is similar to ASAP in that it has pairs of synthetic and performance examples, with 497 distinct music scores paired with 2189 performances, and audio files of each performance. However, unlike ASAP, all files are encoded in MIDI, which makes it impossible for us to parse annotations from them. As such, we developed alternative techniques of extracting annotations from these files alone, as we discuss later in section 4.

3.3.3 Other Datasets

There exist many extensive datasets that we can use for pretraining our models on music generation. The main candidates were the MAESTRO, Lakh MIDI and Giant-MIDI datasets, primarily due to their size.

The first pretraining candidate was the MAESTRO dataset [Hawthorne et al., 2019], this performance-only dataset bears similarities with the ASAP dataset in its genres and composers. This would help ensure the types of pieces the model learns to generate in pretraining are similar to that which it learns the mapping between in the full task. However, the number of performances was not significantly larger than the size of the ASAP dataset, with only 1276 performances, signalling to us that this may also suffer from overfitting during the pretraining process.

The Lakh MIDI dataset [Raffel, 2016] would appear to remedy that problem, with 176,581 examples being enough to train models with proven results like Museformer [Yu et al., 2022] and MuseGAN [Dong et al., 2018]. However, when investigating the types of songs the dataset includes, we see a wide range of genres expanding much beyond our scope of classical piano, such as pop songs like Thriller. This may introduce unwanted complexity with the pretraining process.

Finally, we looked at the Giant-MIDI dataset [Kong et al., 2022], the largest classical piano MIDI dataset to date. Additionally, audio recordings for all pieces are also present. They constructed this dataset by starting with scraped information about each piece, such as the name of the music work and the name of the composer. Using this, they searched for audio recordings on YouTube for accompanying recordings, applied convolutional neural networks over the audio to ensure it was the same piece, then converted the recording into MIDI using an open-source piano transcription system[2]. This resulted in 1237 hours of data.

While smaller than the previous Lakh MIDI dataset, Giant-MIDI is much more focused on the types of performances we hope to emulate as it exclusively focuses on classical piano music, and is still significantly larger than our original ASAP dataset, which should help to mitigate overfitting with the right model design.

[1] https://cheriell.github.io/research/ACPAS_dataset/
[2] https://github.com/bytedance/piano_transcription

We did also consider MuseScore, a scorewriter for Windows, macOS, and Linux supporting a wide variety of file formats and input methods. Through open-source contributions, it has become a leading music notation software used by musicians and composers worldwide, offering a powerful and versatile platform for creating, editing, and sharing sheet music. As we noted earlier, MuseScore has a huge amount of pieces that can be scraped; 1.6 million were available in 2017. These are encoded in MSCZ format, which contains information about both notes and annotations. Converting this format to MusicXML, we could apply our instruction parser to extract the annotations.

However, the average quality of scraped MuseScore examples is low. Through some manual data exploration of randomly-selected samples, the majority of available examples are short, simple pieces. This makes it impractical for most of our needs, as generation models trained on this will be bounded by the low quality of examples.

Preprocessing

Due to the complexity of the task and lack of supporting research, we researched and developed several systems to aid future work in this field. Having reviewed the wider computational music field, we identified research and systems that were required to further progress on our task. We identified data to be the most significant bottleneck; with such little available data, creating effective machine learning models is challenging. We also recognised the need for interpretability tools that would aid the understanding of music for users without significant musical experience.

From these requirements, we devised several systems that would help towards solving them.

4.1 Optical Music Recognition

We developed an instruction parser to extract annotations from MusicXML files, but as we noted few datasets contain these files. Due to this, we pursued an Optical Music Recognition (OMR) approach that would jointly recognise notes, rests, and annotations from an image of sheet music. We can then use the extracted annotations as our instructions for the piece. Images of sheet music are more abundant and easier to find than MusicXML files for a piece of piano music, and thus this tool supports the creation of larger instruction datasets. As far as we know, jointly recognising symbols and annotations has never been attempted in OMR.

4.1.1 Dataset

This task requires taking an image of sheet music as input, and returning the symbols and annotations present in the image as output. We decided to use MuseScore to obtain the dataset for this task as we can gather a large number of images with ground truth labels. It usually contains short and simple examples with few annotations, so we filtered our dataset to only include suitable examples.

We started constructing this dataset by scraping MuseScore, through downloading user submissions to the website. By adapting a publicly-available script[1], we were able to download up to 1.6 million MIDI files. We chose to download 20,000 random files.

To clean and pre-process the data we made use of the MuseScore plugin feature. The MuseScore application allows the creation of custom plugins to manipulate uploaded pieces of music. We found an existing plugin[2] for converting music files en mass, and adapted it for our needs. We expanded on it to add additional preprocessing specific to our dataset[3]. Additionally, we added options to improve functionality and some conversion bugs present in the plugin[4].

Our downloaded files were in MSCZ format, the standard MuseScore file format. We first converted this to MusicXML using the plugin, then ran a custom Python script to remove the author's credits from the file, as we do not want those to be identified as annotations. We then used MuseScore to generate

[1] https://github.com/Xmader/musescore-dataset
[2] https://github.com/Jojo-Schmitz/batch_export
[3] https://github.com/ojaffe/batch_export
[4] https://github.com/ojaffe/Remove-First-Score

multiple images of every piece, with every image cropped to fit one stave. An example is shown in Figure 4.1. Note that only the top stave is showing, as we chose to only include one at a time to simplify the prediction of symbols.

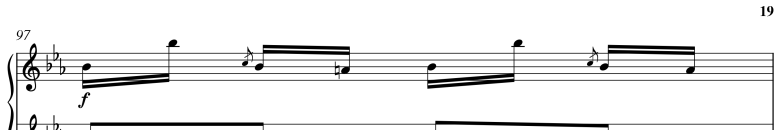


Figure 4.1: Example of a cropped image produced through preprocessing for OMR

We parsed the MusicXML files to extract all symbols and annotations that were present in every image. We then cleaned the data. We first removed images that had no corresponding symbols or annotations, which occurred infrequently as a bug from the conversion process. Then, we removed images that had no symbols or annotations, as often pieces would have long pauses in them. We also removed all images that had the incorrect resolution, which was a result of an inherent bug in the MuseScore application changing the sizes of output scores.

After all the data-cleaning steps, we had 177,213 examples. To ensure we discarded the majority of simple and sparse scores we only selected dense examples with annotations, where dense examples are those that contain many symbols. We then randomly selected 10,000 examples, with the probability of an example being selected proportional to its density and number of annotations. This resulted in a diverse dataset, with the majority of examples containing complex musical structures and annotations.

We release the full unfiltered and uncleaned dataset of 200,000 examples to encourage further research in this field. The dataset contains approximately 6.5GB of images and respective MusicXML files[1]. We also release our cleaning methods[2].

4.1.2 Architecture

Jointly recognising both notes and annotations is challenging as they are very different to one another. A perfect classifier for recognising symbols learns to ignore noise, such as staffs and barlines, and identify symbols of specific shapes. On the other hand, recognising annotations requires distinguishing words from symbols, and words from one another. These require different features, and thus we designed our architecture in such a way that would have a shared base with separate sub-models for both, allowing each sub-model to learn its specific task.

Intuitively, the first portion of the architecture should act as a universal feature extractor. It will reduce the effect of noise, perform preprocessing on the staves, and identify low-level features. These will then be passed to both classification heads, which learn how to combine the features to best suit their task.

We decided to model this as a machine-translation task and use an encoder-decoder architecture, with two separate classification heads for predicting the symbols and text respectively. The intention was to have the encoder learn to extract important features of the piece, and the decoder would act as an autoregressive language model to repeatedly predict what is present in the piece, conditioned on the encoding of the image. Using a language model provides a significant advantage over other approaches, as the model will learn likely sequences of symbols, reducing the probability of catastrophic errors and improving model performance.

We used a convolutional recurrent neural network (CRNN) as our encoder. A convolutional neural network (CNN) transforms the image into a lower-dimensional latent vector, decreasing the computational cost of further processing. This is then fed into a bi-directional LSTM to encode spatial information into the features, which is especially useful for annotations as they can span wide sections of the image. Our

[1] https://drive.google.com/file/d/1dO81DS4cvIMuDXYgIha407P6zhlg4W67/view?usp=share_link
[2] https://github.com/ojaffe/Polyphonic-OMR

decoder is a transformer-decoder, with separate classification heads for symbols and annotations. These heads take the final hidden state of the transformer-decoder, and pass it through a couple of linear layers, then make a prediction based on the logits in the final layer. A visualisation of our architecture is shown in Figure 4.2.

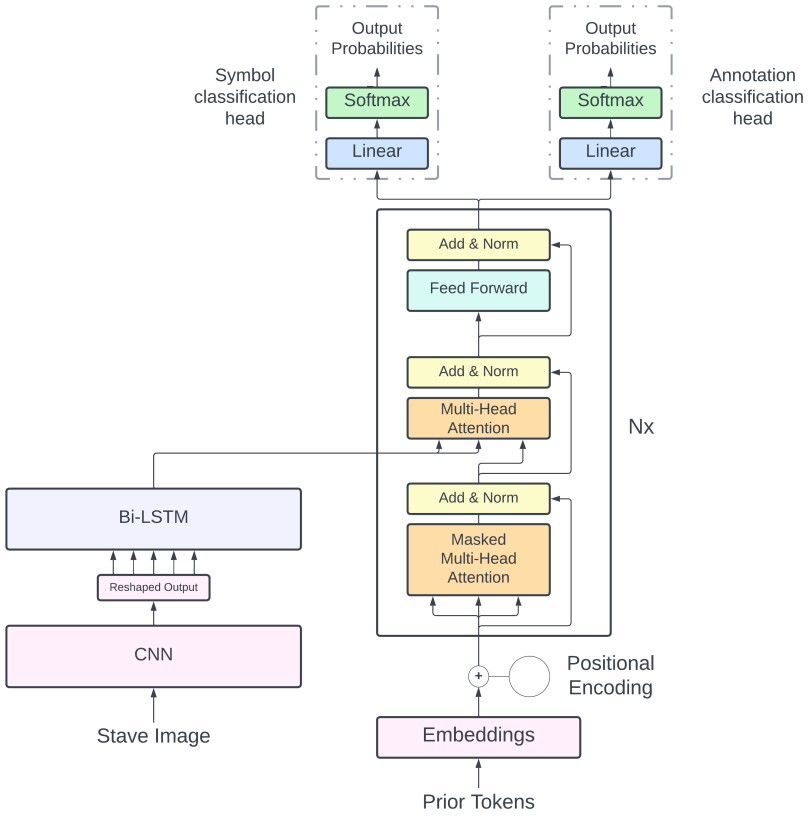


Figure 4.2: Visualised architecture of our OMR model

Due to success in a range of computer vision tasks, we chose to use a residual-18 CNN for our encoder [He et al., 2016]. We used a bi-directional LSTM with 2 layers and a dimensionality of 128. Our transformer had 2 layers, with a dimensionality of 256, 8 heads, and feed-forward dimensionality of 128. We used dropout with $p = 0.1$ and a batch size of $n = 16$ to increase resilience to noise. We also choose to weight our classes and assign more weight to the annotations. Since the number of notes is significantly larger than the number of annotations, models without weighting would focus little on classifying annotations. We take a dataset split of 80, 10, 10, for train, validation, and testing respectively.

4.1.3 Evaluation

There is an ongoing debate about which evaluation metrics are suitable for OMR [Byrd and Simonsen, 2015]. Symbols arguably have different relative importance to one another, the notation isn't standard, and the level at which errors should be counted is unclear. Since we are only concerned with recognising the exact symbols in sheet music, we primarily use Symbol Error Rate (SER) to evaluate the performance of our model. The symbol error rate is defined as the number of editing operations (insertions, deletions, modifications) needed to produce the ground truth from the predicted sequence.

We consider the recognised annotations similar to other symbols and include them in the SER metrics. This allows us to directly measure how including the annotations decrease such metrics.

4.2 Reverse Engineering Tempo

OMR solves some problems with generating a suitable dataset, but has its own issues: particularly, not all sheet music is annotated sufficiently. Additionally, applying the method requires finding datasets of MIDI files accompanied by their respective sheet music, of which there are not many. Using the method thus either requires such a dataset, or a scraper to create one.

As such, we designed an additional system - given a MIDI file of a music piece and a human performance of the same piece, infer the instructions from the human performance. In addition, the MIDI file should reflect the piece without any of the sheet music instructions applied to it. Using such a method only requires a dataset that contains MIDI and performance pairs, which are required to train the main model anyway. Though this is an approach for which there are not many resources or solutions to similar problems to reference (due to the task being niche), there was still an incentive to explore this method more: if it worked well, we would be able to save time in finding additional data.

The approach we designed was to find significant differences between the given MIDI file and performance. A significant difference in this context is a note, or set of notes, that is played in an aurally noticeable different tempo in the performance compared to the MIDI file (this is an abstract notion, so we also had to decide what constitutes as aurally different - this is discussed later in section 4.2.2). Identifying such notes would then allow us to see how different the tempo is and where it is different, and then translate this into the relevant instruction. Given that this difference is simply a duration in time units, the latter part of this task is simple, since all instructions, that correspond to changes in tempo, can be mapped to a duration of time (typically in beats, which can then be converted to time units).

4.2.1 Comparing MIDI files and Performances

First, we needed to design a method to compare MIDI files and performances. Performances are usually recorded in an audio format such as *mp3* or wav, so they cannot be compared directly. Additionally, converting an audio file to a MIDI file is difficult, since the former records signals and the latter records music notes. Whilst techniques do exist, they are not particularly robust and only work with simple files. Since the files we wish to work with may be complicated, this is less attractive than the opposite: converting from MIDI to an audio format.

The most common way to do this is using a synthesiser: an electronic musical instrument that generates audio signals. Simply put, it is an automated system that can play MIDI files. Using a synthesiser allows us to use any MIDI file, rather than relying only on MIDI files that are accompanied by a human performance that have been played exactly to the MIDI file. There are however many synthesisers: as many as there are instruments, since each is made to emulate one. As such, it is important that we use the same synthesiser used in the performances.

Doing so gives us an audio signal, which we can plot with time on the x-axis and amplitude on the y-axis as seen in Figure 4.3.

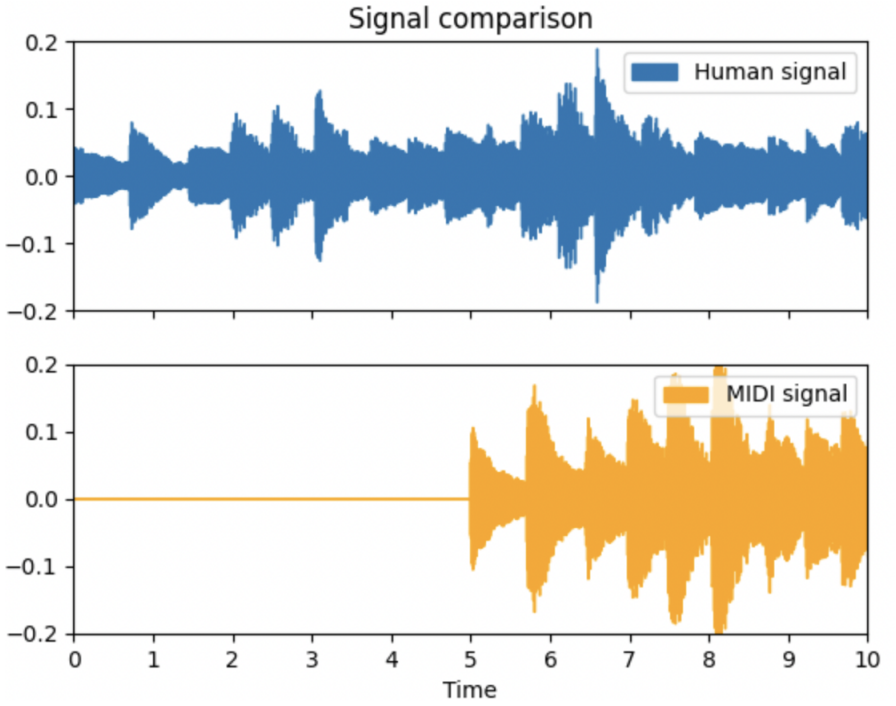


Figure 4.3: Comparison of signals, one of a human performance and one of a synthesised MIDI file, from the piece Chopin's Piano Étude Op.10 No.3

Now that they're both in the same format we can start comparing them. The next challenge was finding the notes in both signals, and comparing the time and amplitude of these notes. For the synthesised MIDI signal, we could use the timing of the notes from the original MIDI file, but we still need to know the times for the notes in the human performance. Naturally, they won't occur at the same time in both files: if they do, this defeats the point, since it means the performance is no different from the score. As such, we need a mapping from the times of the notes in the MIDI to the times of the notes in the performance.

4.2.2 Aligning MIDI and performance signals

Two signals are aligned if the notes in both signals occur at the same time (provided they capture instrument audio). If we take a synthesised MIDI signal and a performance signal that are not aligned, and perform a process to transform one of the signals such that they are aligned, then the newly aligned signal gives us our mapping exactly. Thus, this is the process we used.

There are many ways of doing this, but the approach we adopted was that of Dynamic Time Warping (DTW) [Raffel and Ellis, 2016]. Before discussing this process, it's best to describe the DTW algorithm.

DTW is typically used to compare time-series data - data that measures the change in some property against discrete changes in time. Given two collections of time-series data in the same domain, comparing them is an important task to measure the impact of certain variables. When these are the same size, this is a simple task: an index i in the first collection corresponds to the same index i in the second collection. However, when they are not the same this is not necessarily true; for instance, when data for a few time points is missing or incorrect. We still wish to compare such data, but such an approach will not work. The same is true for our problem: both signals are very unlikely to be the same size; in fact, we expect them to be different.

DTW solves this using dynamic programming, by computing a least-cost matching. To do this, we follow the standard procedure for dynamic programming and design a Bellman optimality equation. For this algorithm the equation is quite simple if we define it for two points in time:

$$DTW[i,j] = \text{cost}(i,j) + \min(DTW[i-1,j], DTW[i,j-1], DTW[i-1,j-1])$$

Therefore, suppose the length of the first signal is n and the second is m, then by computing $DTW[n,m]$ we will get a matching from the beginning to the end of both signals. For simplicity, we can abstractly think of $cost(i,j)$ as the distance between point i in the first signal and point j in the second signal; for our domain, intuitively, the greater the difference in time and amplitude, the larger the cost.

DTW also enforces a key constraint: the matching must be monotonically increasing. If some point x_i from the first signal is matched to some point y_i in the second signal, then for all x_j points such that $x_j > x_i$, the point in the second signal they are matched to, some y_j, must be such that $y_j > y_i$. Particularly for our domain, this is a necessary constraint; no performance of a piece will alter the order of notes played, only the manner in which they are played, so we must enforce this constraint to prevent any such matching from being returned. For the same reason, the first and last index in both signals must be aligned. Finally, due to the potential difference in sizes, we allow multiple indices to map to the same index from the other signal (though the reverse is not true; a single index should only match to one index from the other signal).

To implement this algorithm, we need to implement the cost function. Whilst we have described this as an abstract notion, the next section will discuss this formally.

Defining the cost function for DTW

A cost function for this task should, for two time points, be greater if the two points occur in significantly different parts of the piece (e.g., different notes), and be lesser if the two points occur in similar parts of the piece (e.g., the same note). Initially, we might try simply comparing the amplitude and time difference for two time points. Doing this would require testing every single possible alignment, of which there are many. Rather than doing this, we can somewhat abstract the process into Fourier transforms, which can be done much faster.

A Fourier transform is a mathematical operation that computes the frequency components of a given function. These components, when summed, give us the same function. Typically for time-series analysis, it is simpler to look at these frequency components, and such is the case here. It turns out that if we take the Fourier transforms of two signals and multiply them together, we get a matrix D, where $D[i,j]$ gives us a value that follows the aforementioned specification for the cost function. As such, we can interpret $D[i,j]$ as the distance or cost for matching point i in the first signal to point j in the second signal. From now on, we refer to this matrix as the distance matrix.

However, simply applying the Fourier transform to the entire signal will not yield the result we want. We need to apply it over different sections of the piece. This is because the frequency components for different parts of the piece will be different, as different notes are played in the piece. As such, we discretise the signal into bins, where these bins are sections of the signal. This can be done using the Short-Time Fourier Transform (STFT).

The approach that Raffel and Ellis used employs a type of STFT, called a Constant-Q-Transform (CQT), which has a design suited for musical representation. In the STFT, these bins are typically the same size, but in the CQT these bins are logarithmically spaced. The result of computing a CQT on a signal is called a Constant-Q-gram (CQ-gram).

Using the approach described earlier, we can compute a CQ-gram for both signals, multiply these CQ-grams together and use the resulting distance matrix to implement DTW.

Using alignment to find significant differences

When we compute the alignment using DTW, we get a mapping from the synthesised MIDI signal to the performance signal. This mapping is from a set of indices from the synthesised MIDI signal (that can be interpreted as time points in the signal) to another set of indices from the performance signal. We can use this mapping to transform the original MIDI file into a new, aligned, MIDI file.

There are many ways of finding significant differences with this. The most obvious idea is to compare the aligned MIDI file to the original MIDI file. This is convenient, since both should contain exactly the same notes, and so we can compare the position of a note in one MIDI to the position of the same note in the other. We did try this, but the main problem was that this was not necessarily true. The number of notes post-alignment was, seemingly arbitrarily, sometimes less than the original number of notes. We suspected the issue to be with the library that computes this transformation, but even after inspecting the source code we could not find the issue.

Instead, we opted to use the indices and the mapping. We can look at an index of one signal, and the position of the index it is mapped to. However, we cannot simply compare these values, as there might be an offset. Instead, we compute the difference of differences: for a given array, the difference of differences is an array of the difference between consecutive values. This essentially tells us the time between points in the signal, which is a more intuitive measure of tempo: for two points in the first signal, x_i and x_j, that are mapped to two points in the second signal, y_i and y_j, if $|x_i - x_j| < |y_i - y_j|$, then there is less time between those points in the first signal, indicating a higher tempo in the first signal. This makes our task of interpreting instructions simpler as well, as we can directly map the difference size to an instruction.

In the difference of differences, we can think of any value greater than some threshold as significant. We then need to decide what constitutes a significant difference. This is difficult, as it is not clear how to select this value; setting it too high will cause all significant differences to be lost, and setting it too low will cause everything to appear as a significant difference. We also can not ask the person where they made significant differences; even if we could, it is unlikely they would know with enough precision. That said, one method may be to simply take the median over a number of performance and score pairs, and use this median to set the threshold. This provides an approximation based on real performances, which is perhaps the best one can hope for in finding a suitable value for this threshold.

It is worth mentioning now that, indeed, one can simply do this with two MIDI files directly, without the need of dealing with Fourier analysis and the DTW algorithm. The above described approach, of taking the difference of differences, can be used between the two MIDI files. However, a mapping is still needed, as it is unlikely the two MIDI files line up perfectly; for instance, even for the same piece, a score and performance may be different if there are any mistakes in the performance. The difference of differences is only valid between two sequences of the same length. Performing DTW between both MIDI files is an option, but if the performance is not in MIDI format it requires it to be converted to MIDI format, and conversion may introduce errors or noise in the piece. This approach is more general, and allows for data in either format to be treated the same.

4.3 Reverse Engineering Dynamics

Dynamics is the other important aspect to improve the realism of a piece of music. Just like getting tempo from audio, a similar idea can be applied to obtain dynamics information, and extract instructions related to dynamics.

Taking the piano as an example, dynamics can be changed by applying a different amount of force on the keyboard. The increase in force might induce other side effects, such as creating resonance, which can make neighbouring strings vibrates together and introduce a metallic feeling. Analysing the exact sound signature of each notes on the musical instrument at varying dynamics is considered beyond the

scope of our objectives, as it requires more complex knowledge to create physics modelling and simulation. Therefore, in reality, change in dynamics affects many sound properties other than just volume, but we simplify dynamics as volume for the scope of this project.

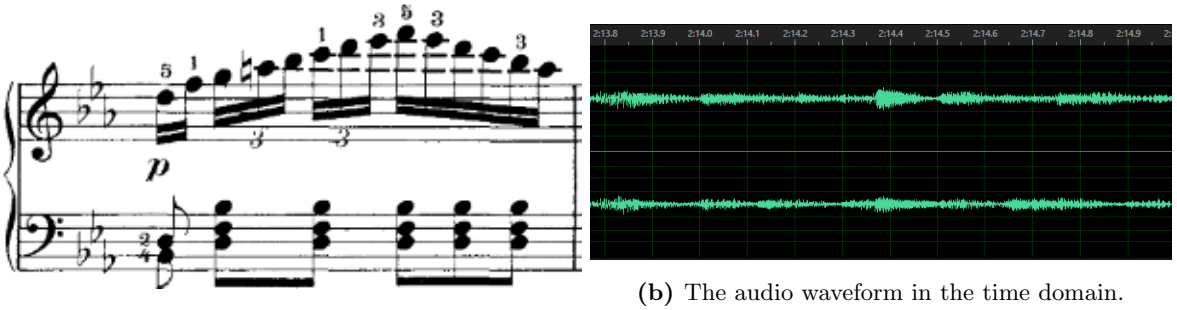
4.3.1 Volume Extraction from Audio

The dynamics extraction starts by reading volume information from audio. Any audio is essentially a collection of different waveforms. These waves vary in amplitude, i.e., they oscillate as time passes. Thus, there is a simple approach, which is to read the volume by finding the average amplitude at every time step. We can then compare this with a performance audio recording to find differences. Nevertheless, this can cause some problems in practice due to the varying recording quality of the audio, most notably noise in the audio signal.

To reduce the impact of noise, filtering needs to be applied first. Noise is considered to be random, and can span across the entire recording. Consequently, the unpredictability nature of noise means that it is not practical to remove noise when working in the time domain. It is observed that, however, noise generally follows a probability distribution; for instance, white noise follows from a uniform distribution. Although the distribution does not necessarily help in predicting when a noise signal appears, it allows us to know how often it appears. As mentioned earlier, the Fourier transform can be used to convert signals to the frequency domain. This makes finding noise easier, and allows noise to be removed by decreasing the amplitude of signals at particular frequency bands.

There are many variations of the Fourier transform, as mentioned earlier, like STFT and CQT. STFT computes the frequency domain of a time series signal and outputs a 2D matrix of frequency against time, and the value in each matrix cell represents the amplitude. In contrast, CQT discretises continuous frequencies into a number of pitches and outputs a 2D matrix of pitch-time. The main difference between frequency and pitch is that frequency specifies the number of times a signal waveform oscillates in a second, while pitch represents the perceived fundamental frequencies (explained in detail in section 4.3.1) of a sound wave.

The frequency domain brings a few additional benefits compared to the time domain. As seen in Figure 4.4, it is difficult to search for particular melodies in time domain representation, since different music notes are distinguished by their frequencies.



(a) Sheet music score

(b) The audio waveform in the time domain.

Figure 4.4: A measure from Beethoven's piano sonata, Op.81a, Movement 3

In contrast, working in the frequency domain allows analysis of notes in a given piece of music. As shown in Figure 4.5, a clear trend of note progression can be observed highlighted by the arrows.

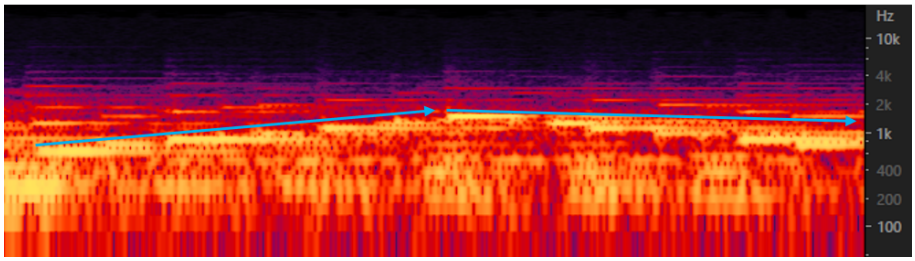


Figure 4.5: The STFT frequency spectrum of the same section in Figure 4.4; brighter colour has higher amplitude; frequency is displayed on the right.

Due to the fact that there are only a limited number of notes on a musical instrument, one can argue that continuous frequency not only contains excessive information that makes analysis of music signals unnecessarily difficult, but also it is not sufficiently robust against noise because not all frequencies are contributed by music notes. The pitch spectrum, however, reduces the dimensionality of y-axis to save memory and computing power for signal processing, by removing frequencies that do not belong to any music note, so that it enables easier analysis of music harmonic structure and tonality for later tasks. By utilising CQT, continuous frequencies are binned into a number of fundamental frequencies, as illustrated in Figure 4.6, where the label of y-axis is replaced with discrete frequencies of each music note, rather than the continuous frequency in Hertz. It can be shown that Figure 4.6 shows clearer music notes than Figure 4.5.

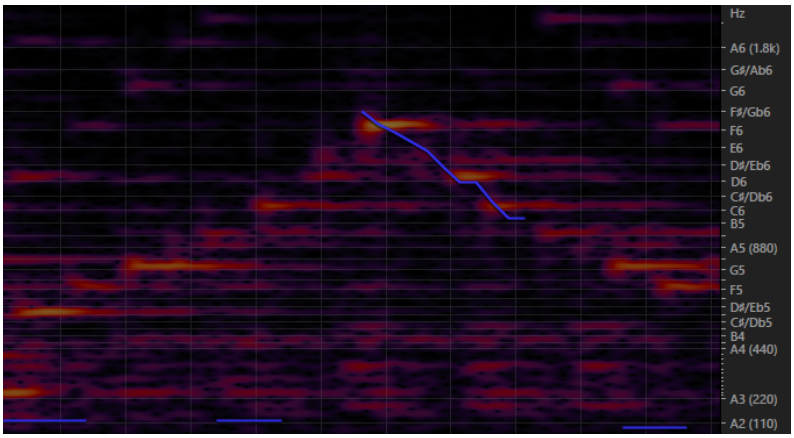


Figure 4.6: The CQT pitch spectrum of the same section in Figure 4.4; fundamental frequencies are displayed on the right.

Fundamental Frequencies

A simple signal can be modelled with as sinusoidal wave $s_k(t) = A\sin((k+1)\omega t + \phi)$, where A is the amplitude, ω is the frequency and ϕ is the phase. Then, $k \in \mathbb{Z}^0$ is the harmonic number; when $k = 0$, this frequency is referred as the fundamental frequency, and all signals with $k > 0$ are the harmonic frequencies of this fundamental frequency. The set of signals $\forall k \in \mathbb{Z}^0, s_k(t)$ forms the harmonic sequence.

In a musical instrument, the fundamental frequencies of each note is defined based on the tuning system. A tuning system specifies the number of note in an octave, as well as the frequency ratio between adjacent notes. In this project, we only focus on a tuning system called equal-temperament with note $A_4 = 440$Hz, and it defines 12 fundamental frequencies in an octave, where the ratio between each

adjacent note is equal, i.e., $\sqrt{12}$. Even though this tuning system does not sound as natural as some of the more ancient tuning systems, it is the most widely used arrangement due to its simplicity for manufacturing and tuning.

Volume to Dynamic Mapping

When working with a musical instrument, the sound signal from a single music note is not a single wave, but a sum of many fundamental frequencies and their harmonics, while the fundamental frequency corresponding to this note has the highest amplitude. For example, the note A_4 shows multiple peaks in the CQT spectrum, including seemingly unrelated frequencies like E_6 and $C\#_7$, within which 440Hz has the highest peak. This imposes a significant challenge to determine the label of a given sound wave for an instrument, as different music notes can have vastly different frequency signatures. As a simplified model, we first attempt volume to dynamics conversion by finding the average volume at every time step.

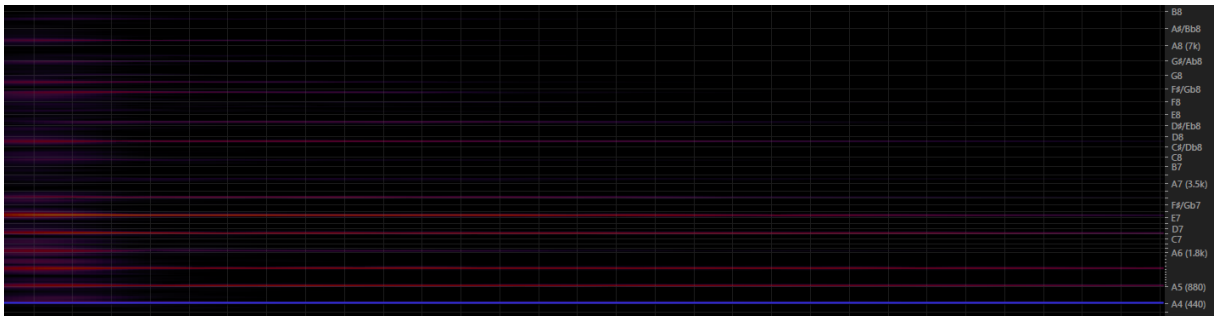


Figure 4.7: The CQT spectrum of an A_4 note

After obtaining the CQT spectrum, volume data can be looked up directly. Adaptive thresholding is first applied to remove signals that are very quiet, as they are potentially harmonic frequencies to simplify the spectrum. Choosing the right value for the threshold can significantly impact the accuracy of volume to dynamics conversion, as it might accidentally mute fundamental frequencies if the threshold is too high, or not clean up harmonics properly if it is too low.

Followed by this step, the mean volume at each time step is determined by calculating the mean amplitude among all non-zero-amplitude fundamental frequencies found by CQT. This operation returns an array. The array is finally cleaned up by normalising all values so that they sum up to one.

To convert the volume to dynamics that is compatible with MIDI file, such that each array element should be within closed range from 0 to 127, a constant dynamics factor can be used to scale the array, and convert the data to an integer. This newly obtained array contains converted dynamics information at every time step in the audio recording. Finally, it is necessary to determine a mapping function that maps from audio time to MIDI time so dynamics can be modified in MIDI file correctly, which can be achieved by reusing results from MIDI alignment. The alignment data provides information that, for a given playback time in the MIDI file, returns MIDI notes that are active, and dynamics adjustment can be applied to such notes.

4.3.2 Signal Decomposition

The initial attempt for reverse engineering dynamics by applying dynamics changes based on a simple mean volume at a time step works for music pieces with simple melodies. Meanwhile, we are looking for a solution for classical music with variable complexity. Although the current result shows a variation of dynamics as playback time passes, if we look at each note at the same time step, all notes are assigned with the same dynamic. This does not make any difference if there is only one note per time step, or in case of multiple notes presence, they are played with the same dynamic.

In a majority of cases, music can have a mixture of notes with different dynamics at any time. In a piece of expressive polyphonic music, for instance, it will typically mix melodies with accompaniment with varying dynamics. In the example shown in Figure 4.8, it can be seen that notes at the top corresponds to the melodies that have a brighter colour, and thus higher amplitude, than notes at the bottom. Setting all notes in a time period with equal dynamics will no longer be suitable in this scenario.

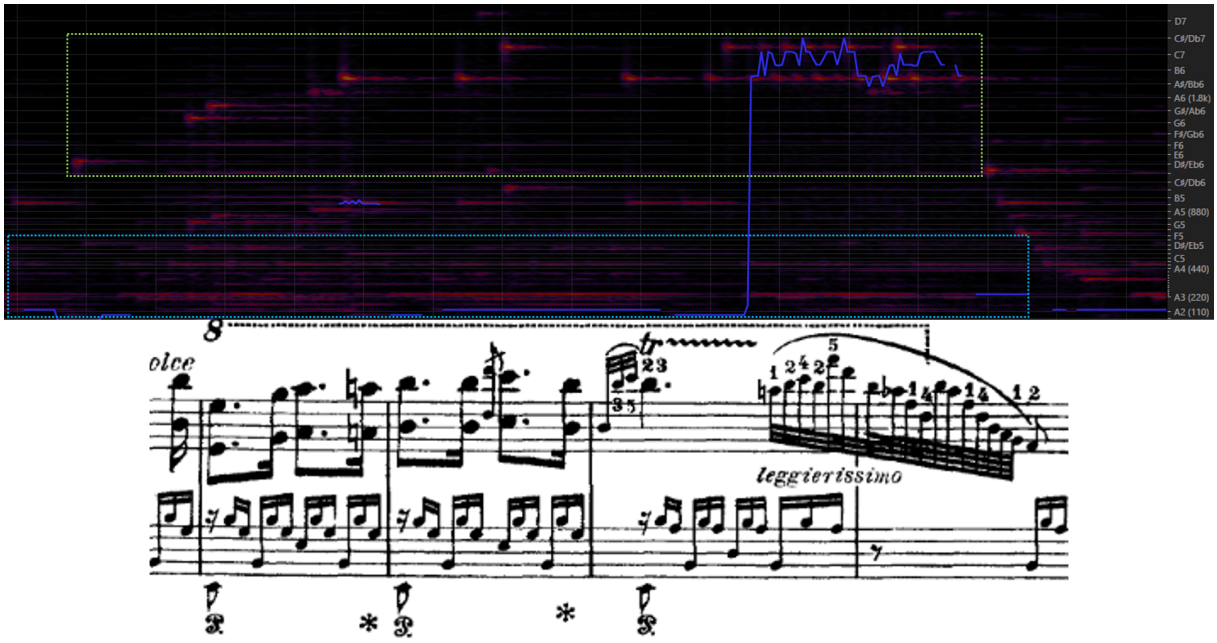


Figure 4.8: An example of voicing in music from a section of Liszt's S.162, No.3, where the right-hand notes (in the green box) are noticeably louder than the left-hand notes (in the blue box).

Therefore, a better approach to determine the dynamics for each individual note at a time step should be used to produce a more realistic result. The problem of signal decomposition can be formulated as to decompose a composite musical signal into fundamental frequencies of each individual music note, and determine the weight of fundamental frequencies, the weight can then be used to infer their dynamics.

Generalised Signal Decomposition

Signal decomposition is a multi-disciplinary subject across many areas. As mentioned previously, different music notes are distinguished by their frequencies, such that it is common to perform signal decomposition in the frequency domain.

General approaches for signal decomposition involve the use of a multiple band-pass filter, and use algorithms to determine the frequency bands for these filters [Eriksen and Rehman, 2022]. This generalised approach is most popular in engineering for detecting noisy sensor signals and can be extended to other areas, such as seismic detection and medical scanning.

A band-pass filter is a frequency domain filter, that defines a lower and upper frequency bound where frequencies are allowed to be passed through unmodified, all other frequencies outside this bound will be suppressed.

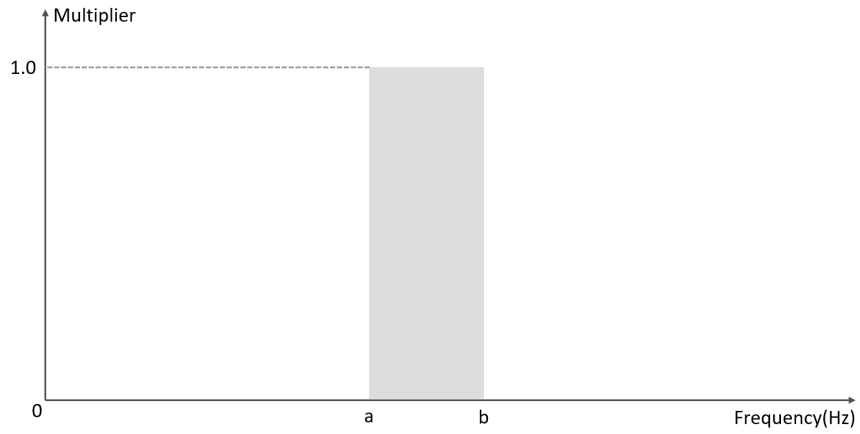


Figure 4.9: A generalised model of a band-pass filter, with frequency band a to b.

Results from [Eriksen and Rehman, 2023] comparative survey show that this type of filter and algorithms for determining this frequency band is ideal for signals with a simple and limited range of frequencies; particularly for signals with frequencies that do not overlap with each other. This is useful for applications such as recovering sensor signals from a noisy environment, but does not perform well for more complex musical signals, which are highly varied in nature. The example in Figure 4.7 shows wide-band overlapping frequencies for only a single music note.

Fundamental Frequency Estimation for Music Signals

Fundamental frequency estimation, also known as F0-estimation, was a technique developed originally for speech recognition, and later was extended to other areas, such as estimating music notes given an audio recording.

F0-estimation can be divided into 2 categories based on application [Klapuri, 2003]. Predominant F0-estimation considers one note that is the most significant in a given section of the signal, even in the presence of other harmonics. This method achieves promising accuracy and is widely used in speech recognition, as only the speaker's voice is required. Nevertheless, the underlying problem for analysing a music recording is the accurate handling of polyphonic music in a setting with multiple notes. Multiple F0-estimation, on the other hand, provides a complementary approach to this problem, which allows detection in the presence of multiple notes.

Multiple F0-estimation has been actively researched over the last few decades, and a large variety of approaches are available. The error rate of estimation goes up as the polyphony increases, and most methods achieve an accuracy of about 50 percent when the number of polyphony gets to 6 [Bittner et al., 2018, Yeh et al., 2010]; one model achieves a significantly better accuracy compared to others, of more than 80 percent [Klapuri, 2003]; however, it is still below the accuracy expected in this project. The main challenge of multiple F0-estimation is the unpredictable nature of sound, such as acoustic echos, and instrumental effects like piano pedals, which can all interfere with the overall accuracy of estimation.

Recently, the use of neural networks for F0-estimation has started to emerge. In spite of the fact that a number of models demonstrate their superior estimation accuracy [Singh et al., 2021], taking this approach requires the use of a special pitch tracking dataset in a format that is different from our current ones. Due to limited development time, it is better to constrain the complexity by not using any deep learning approach for this particular problem, which would add complexity.

4.3.3 Alignment-based Signal Decomposition

The approach taken to solve the problem of signal decomposition in this project is slightly adapted from the solution used in reverse engineering tempo. Recall that DTW is an algorithm to align MIDI

and audio signals, which acts as a mapping table between MIDI and audio time domain; the ultimate objective in the reverse engineering dynamics problem is to extract dynamics from audio and apply to MIDI. In this case, instead of estimating fundamental frequencies solely from audio, a MIDI file can be used in conjunction with that to simplify the problem. Whilst this is straightforward, the major drawback of this alignment-based approach is the reliance on DTW, and the accuracy is heavily affected by the performance of DTW.

The algorithm is performed by iterating through every note in the input MIDI file. For each of these notes, considering its start time as the MIDI time, we can find the audio time using the alignment data. This audio time can then be used to locate the time step in the CQT spectrum, where each time step contains a number of bins, and each bin corresponds to a musical note, whose volume information can then be looked up directly based on the pitch of the note.

The last step is to provide a mapping from volume to dynamic. As the volume is an unsigned normalised value (i.e., a fractional number in the range $[0.0, 1.0]$), a velocity curve can be used to map this value to the standard MIDI velocity value, which is an integer in range $[0, 128)$.

Velocity Curve

The choice of velocity curve can impact the dynamic texture of the output MIDI file. For $v \in [0.0, 1.0]$ as volume input and $d \in \{x | x \in [0, 128)\} \cap \mathbb{Z}^0$ as dynamic output, a simple full-range linear mapping below might be used.

$$d = 128v$$

A linear mapping might cause the problem of over-amplification or over-suppression of a note if the alignment gives an inaccurate mapping. A more realistic velocity curve can be used to shrink the output range by tweaking the factor α and β in the equation below, so a note will not be too loud or too quiet.

$$d = 128\alpha v + \beta$$

The velocity curve can also be non-linear, which allows for simulating a different style of instrument. For classical piano, it is typical to set the exponent ϵ to be slightly greater than 1.0 to make it sound smooth.

$$d = 128v^{\epsilon}$$

4.4 Sectioning

Underlying structures are present in most musical compositions; whether it be simple Verse-Chorus-Verse-Chorus repetitions prevalent in modern pop music or Sonata forms seen in classical pieces, some sort of structure exists in most pieces of music. In the design of our system, we wanted to ensure applying instructions was accessible to users with varying musical knowledge. Our intuition was that if we can identify the underlying structure of the piece of music we are given, we can present the user with a clear and simple breakdown of different sections of a piece such that they can use different sections as markers to describe where they would like to apply an instruction.

4.4.1 Clustering

Initially we planned to make use of unsupervised clustering to identify similarities between small time windows within a piece. We found existing works estimating the similarity of MIDI files with various techniques, ranging from text processing [Liumei et al., 2021] to extracting features such as chord

trajectory [Wang and Haque, 2017]. We also found work detailing an approach to embed the MIDI format into a vector space using a Graph representation of the notes and then using a random walk method to encode that graph as a high dimensional vector [Lisena et al., 2022].

However, what we observed with all this previous work was that the focus was on entire pieces as data points, using clustering to find metadata about a piece such as a genre or composers. For our task we would need our chosen approach to work with very short (and potentially overlapping) sections of a piece and be able to distinguish between data points with less drastic differences than two separate pieces of music. What we found, especially with the vector embedding approach was that when given short sections of a piece we were unable to repeatedly find similar sections of a piece due to the lack of overall information.

4.4.2 Dynamic Texture Modelling

We then discovered some work which focused on the same task we were tackling, identifying distinct sections of a piece of music but instead working in the audio space instead of with the MIDI format. Obviously, as a music format converting MIDI to an audio format was a simple task, allowing us to work with these more relevant existing techniques.

Specifically, a paper describing how audio can be modelled as a mixture of 'dynamic textures' [Barrington et al., 2010] was the basis for our sectioning approach. By using an audio file, we can make use of different feature representations such as MFCCs (Mel-frequency cepstral coefficients) or chroma features which we can model with a dynamic texture. A dynamic texture is a method of modelling time series data, like the aforementioned feature representations, making use of 2 random variables encoding the state at time t and also higher-level evolutionary dynamics over previous time steps. This model allows us to represent simple sequences, but by combining them into a mixed model (a dynamic texture mixture or DTM) we can find also represent more complex data such as pieces of music more accurately.

This technique is similar in concept to that of a Gaussian mixture model (GMM) which are used to more accurately model clusters of data points with more general shapes. Where in this case, we combine multiple dynamic textures to model a more complex class of time-series data instead of using multiple Gaussian distributions to model the distribution of points in a space.

If we have a mixture of n components, the i^{th} component is defined by its parameters Θ_i, and we have a random variable z which defines the probability of a given dynamic texture being used for the model. Then in another parallel to GMMs we use an expectation maximisation algorithm to estimate these parameters Θ_i[1]. With the optimal parameters found, extracting labels (j) for each time step in the piece of music is as simple as determining which dynamic texture has the highest posterior probability of representing it:

$$j^* = \underset{j}{\text{argmax}} \frac{\alpha_j \cdot p\left(y^{(i)}|\Theta_j\right)}{\sum_{k=1}^{n} \alpha_k \cdot p\left(y^{(i)}|\Theta_k\right)}$$

where $p\left(y^{(i)}|\Theta_j\right)$ is the likelihood of sequence sequence $y^{(i)}$ occurring under the dynamic mixture defined by Θ_j.

4.4.3 Tuning the Approach

In [Barrington et al., 2010] results they set out the hyperparamaters and steps taken to find the best segmentations for their dataset. Their dataset consisted of a wide range of genres including pop, rock and others that included vocals and a wide range of instruments. In contrast, we limited our scope to focusing on classical piano pieces which we found to work better under different parameters.

[1] For further details on dynamic textures and the parameter estimation process see 'Modelling Music as a Dynamic Texture' [Barrington et al., 2010]

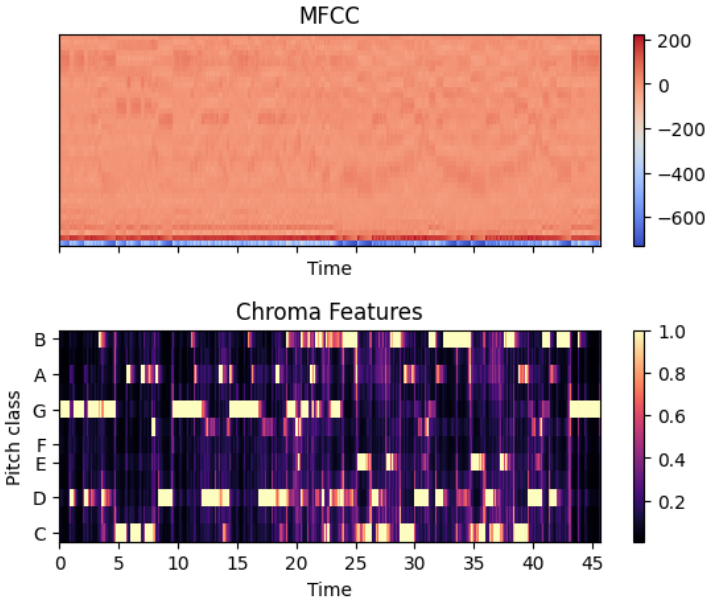


Figure 4.10: The differences between MFCC and Chroma feature representations.

The biggest difference we found was that the paper suggested use of MFCCs as a feature representation, an audio feature representation designed for use in speech recognition. We found that by changing the implementation to use chroma features - a more general audio feature representation that decomposes the audio into the pitches that make up a signal - we got much clearer boundaries between sections and less noisy labelings that switched between two labels rapidly. We believe this is because of the prevalence of vocal performances in their dataset, clear changes in the vocals would be picked up more clearly by the MFCC features, and in contrast to our piano focused dataset, chroma features encode a much clearer picture of how the piece evolves.

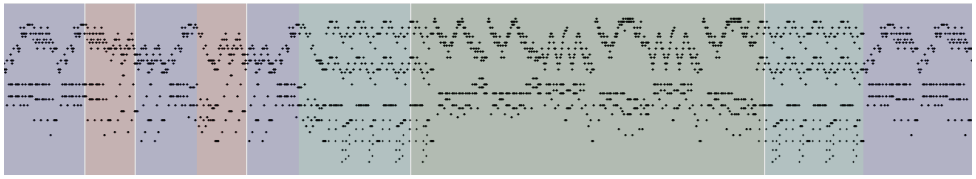


Figure 4.11: An example sectioning of a piece into 4 sections overlaid onto the MIDI representation of the piece.

4.5 Music Instruction Parsing

As part of the humanisation system, we wanted the user to be able to control the way the music is played, beyond simply being able to make it sound human. To do so, we need to allow the user to express how they wish the piece to be played. By analysing this user input, we can modify the playback of the piece accordingly. This can be given to one of the machine learning models discussed in section 5.

This section proposes a more practical approach for integrating user modifiers into the particular humanisation model used in this project. As the humanisation model consumes a sequence of tokens formatted from MIDI music as input, the idea is by formatting user modifiers into such tokens as well and inserting them in appropriate locations in the token sequence, to mimic as if the user modifiers are from the input music itself.

To make the specification of such user modifiers in a more human-friendly way, it is planned to allow the user to input such parameters as a series of sentences containing music instructions written in natural English. These music instructions mainly include tempo instructions (e.g., *Andante Cantabile*, *Accelerando*) and dynamics instructions (e.g., *Forte*, *Crescendo*).

4.5.1 Corpus of Music Instructions

Although it is possible to reuse an existing NLP model and adapt it for this project, the process of training still requires a collection of specialised music vocabulary to build a dictionary that will be used as dataset. This dictionary is typically referred to as a corpus, and the capability of an NLP model depends highly on the range of vocabulary in this corpus.

It is recognised that the number of music instruction is relatively limited compared to language used in daily life, such that these data are gathered manually from a variety of classical sheet music[1]. As mentioned earlier, the corpus only contains tempo and dynamics instructions.

4.5.2 Parsing Pipeline

spaCy[2] is an industrial standard, multilingual, general-purpose NLP library that provides a highly configurable pipeline which can be adapted for different applications. The default pipeline contains a number of commonly used NLP processing steps as illustrated in Figure 4.12.

Figure 4.12: The default processing pipeline provided by *spaCy*; see section 2.4 for information regarding each step

Due to the fact that the built-in model is trained for the application of general language information, it does not produce accurate results for music application. The entity linking step, which is responsible for the disambiguation of synonyms, gives a high error rate that may potentially misclassify certain terms. For instance, *A tempo*, which stands for *return to original speed*, might be broken down into two tokens, and the word *A* may be considered as an *indefinite article*. This is mainly caused by the difference in language, as music instructions are mostly in Italian rather than English. Moreover, the second-most error-prone step is the NER, which has not been trained to consume any music vocabulary and fails to recognise any music instruction.

[1] https://imslp.org/
[2] https://github.com/explosion/spaCy

Specialised Pipeline

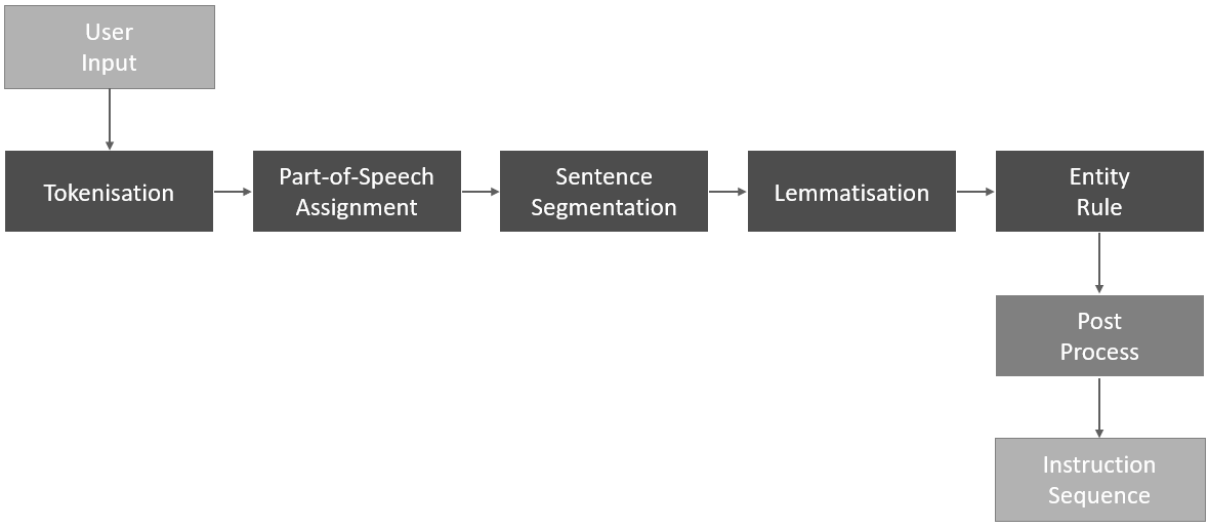


Figure 4.13: The NLP pipeline architecture adapted for application of music instruction parsing

Therefore, certain modifications are implemented and the following pipeline is used for parsing music instructions. Most notably, the entity linking is replaced by a sentence segmentation step, which allows the user to specify all instructions in a single input, provided different instructions are specified in different sentences, and they will be processed independently.

Then, NER is replaced by an entity ruler. Classification of special music terminologies is done in a simple search-based method using the provided music instruction corpus. The search-based approach categorises an entity if an exact match is found. Although this will force the user to specify instructions in the exact same way as they are defined in the corpus, these instructions are standardised conventions on sheet music, so they should always be specified in the same way regardless.

The output from the entity ruler will be an internal data structure of *spaCy*, and an extra step is added to format them into a sequence of instruction tokens that can be passed to the humanisation model.

Language Standard

The language standard defines the format for which users should follow when entering their instructions. There are different types of instruction, some are said to be *monotonic* because the changes take effect immediately; for instance, *allegro* instructs that, from this point towards the rest of the piece, unless otherwise specified, should be played in such tempo (usually *allegro* is around 120 BPM). Nevertheless, there exist instructions that take effect over time; this includes instructions like *ritardando*, which instructs the musician to slow down gradually. For such instructions, it is required to define a start and end time, as well as how much to be changed.

Although introducing extra flexibility to how the instructions can be specified by the users, can be easily integrated into the current NLP parsing model by adding rules to the entity ruler processing step, the major problem is whether the main humanisation model is able to recognise complex instructions. For this consideration, it is decided that the user should not be allowed to specify surplus information other than the instruction itself.

To allow dealing with *non-monotonic* instructions without giving a range of time, the result from sectioning (see section 4.4) can be used to limit the range of effect an instruction can make. As a reminder, the sectioning model segments the music into a number of sections based on the similarity of the melodies;

each section defines the start and end time of the music. Furthermore, users are given choices to adjust the number of sections, providing them a more fine-grain level of control.

To sum up, instructions are divided into two categories, tempo and dynamic; each instruction is paired with a section indicated by a unique section number. A section may have no user-specified instruction, or if provided, may be a collection of tempo and dynamics instructions. Note that some instructions can be combined together; for example *poco a poco* and *adagio* can be combined together to indicate that tempo is gradually converging to *adagio*, which is about 60 BPM.

Matching Rule

The definition of rules in the entity rule step is vital for the accuracy of the NLP pipeline. User input is first processed into a sequence of tokens; followed by a part-of-speech assignment step, which identifies the category of each word. This step allows filtering out information that is not related to the section number and instructions to reduce the number of tokens.

The sentence segmentation then divides and groups tokens by sentence. Although more sophisticated algorithm exists to achieve this task, for example, by performing semantic analysis on the sentence; testing shows that the pretrained model has a high false-positive rate for our usage, possibly due to the fact that the model was not trained for this particular purpose, therefore the method adapted in this project is rather simple, by using full-stop character as sentence separator.

To identify section numbers, a rule-based approach can be used to find the word *section* followed by a numeric character (0 to 9) within a sentence. This can bring a problem of overburdening the user if multiple sections are specified within a sentence, and every section number needs to be pre-pended with the word *section* (i.e., *section 1, section 2 and section 3* rather than *section 1, 2 and 3*). Thus, the algorithm is simplified to assume that all numbers that appear in a sentence are sections, and all matching numeric characters are treated as section numbers.

To identify instructions and classify instructions into tempo and dynamic, the corpus can be used as a lookup table to label the tokens. If multiple tokens of the same type of instruction are found, it is assumed to be a combination of instructions (recall from the previous example, *poco a poco* and *adagio*), such that they will be concatenated into one token in the order they are specified in the sentence.

Finally, instructions need to be mapped onto each section. Denote S, I_t and I_d as a set of sections, tempo instructions and dynamics instructions identified within a sentence; informally, the output is computed by permuting each section with pair of tempo and dynamics instruction; formally:

$$S \times (I_t \cup I_d)$$

Music Generation

This section details the various approaches we tried in developing a music generation model for this task. Our models were trained on both Google Cloud[1] and GPUs available at the University of Warwick. We developed in PyTorch[2] as it offers more features than alternatives for efficiently training large models, such as flash attention. A generic training harness was built[3] to support parsing of datasets and training of arbitrary models.

5.1 Generation as a Translation Task

As we noted when defining the task in section 1.1, music humanisation can be thought of as a music translation task. The sequences (for the score and performance) lie on different distributions, and the instructions indicate a mapping between them. It is noteworthy that their distributions are very similar as the humanised performances are based on synthetic scores. In a language analogy, they could be thought of as dialects of one another.

As we can frame our task in this way, we followed an encoder-decoder paradigm for designing these architectures. In this paradigm, the network has two defining sections: the encoder which creates a high-dimensional embedding of the source sequence, and the decoder which produces the target sequence conditioned on the encoding.

Due to the lack of suitable data to directly train models on translation, we had to pretrain on standard music generation, for which there is significantly more data. To ensure our models learn a relevant task whilst not preventing future fine-tuning, we pretrain only the decoder of the model on music generation. The cross-attention module is disabled to prevent gradient updates by changing the weights in the encoder. To ensure the model generalises well and is malleable to fine-tuning, we employed a number of strategies to prevent the model from overfitting. During pretraining, we can then enable the cross-attention module and significantly increase the learning rate to coerce the model to learn the translation.

A visualisation of this architecture with a transformer-based encoder-decoder is shown in Figure 5.1.

[1] https://cloud.google.com/
[2] https://pytorch.org/
[3] https://github.com/ojaffe/MusicPerformance/tree/c809c75d9edb89b5ac3b8734274e2af4185a747b

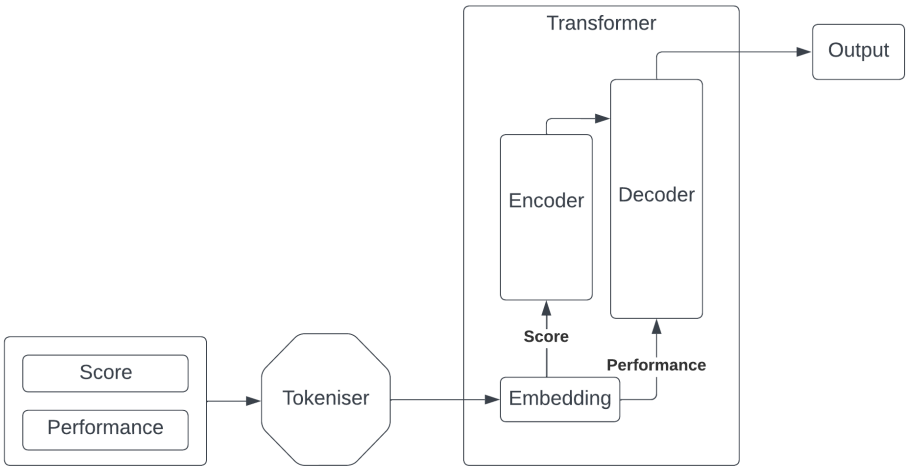


Figure 5.1: The architecture overview of our generation by translation transformer model. For details on encoder-decoder block internal structure see section 2.2.2

Following our analysis of the impact on sequence lengths of different tokenisation methods in section 3.1, we decided to use the octuple tokenisation for all our models. Since each characteristic of a note, such as the pitch or tempo, is embedded independently and then concatenated, we are free to change the embedded dimensionality of the characteristics as long as they sum to the desired total dimensionality. Because we restricted ourselves to only using classical piano, the instrument characteristic is almost redundant as only one instrument is ever encountered. We decrease the embedded dimensionality of the instrument from the default 128 to 4, then split the extra dimensionality between pitch and position as those are the most important for modelling music.

Due to the nature of the task, we had to input the full sequence of both the synthetic scores and performances. Many machine translation based approaches can chunk their sequences, for example, splitting a paragraph into sentences, and then only translating each sentence at a time. However music can not be so easily chunked; the score and performance sequences often had significantly different lengths due to tempo differences, and identifying corresponding sections is a significant task. Applying the sectioning that we discussed in section 4.4 was insufficient, as that method requires setting the number of chunks beforehand, which we would not know prior to manually analysing the piece.

In the following section, we will describe each of the translation-based models we implemented.

5.1.1 Custom Transformer

Our objective for our first model was to build a model from the ground up that could be trained on an NVIDIA T4 GPU[1], as not only would this prove the efficiency of such a model, but would also encourage further research in this field by providing an accessible solution. We started by expanding on our generic training harness for our specific datasets and evaluation metrics. We also created inference methods to make the model generate music from a given synthetic score and instructions, using beam-search decoding.

Architecture

We started by taking a PyTorch implementation of a vanilla transformer[2]. The residual connections and layer normalization present in transformers prevent common challenges in deep learning, which

[1] https://www.nvidia.com/en-gb/data-center/tesla-t4/
[2] https://github.com/hyunwoongko/transformer

encouraged us to use this architecture. Residual connections allow for the smooth flow of information through the network by providing a direct path for the gradient to flow backwards, which enables the network to learn complex representations. On the other hand, layer normalization helps to reduce the internal covariate shift by normalizing the input to each layer, which stabilises the learning process and speeds up convergence [Ba et al., 2016].

For both the encoder and decoder, we used 6 layers, 8 heads, a model dimensionality of 256, and a feed-forward dimensionality of 128. We chose these hyperparameters by evaluating similar models in music generation, and we found these to give good performance without overfitting. We took a split of 80, 10, 10, for our training, validation, and testing datasets respectively. We chose to use adam as our optimizer, as it is a popular method to achieve fast and efficient convergence during training [Kingma and Ba, 2014].

Since we wanted the decoder to predict tokens from a discrete vocabulary, we decided on using cross-entropy (CE) loss as our loss function. Specifically, we implemented a 'smooth' version of CE loss, as proposed by [Szegedy et al., 2016]. Smooth cross-entropy loss adds a smoothing term to prevent overfitting and improve the stability of the training process. It works by replacing the hard one-hot encoding labels with a smoothed distribution that assigns a small probability to incorrect classes, effectively training the model to be less absolute in its predictions. This helps the model generalize better to unseen data and reduces the impact of outliers during training. We choose a value of 0.1 as the hyperparameter to indicate the amount of smoothing, which we found to be a balanced trade-off between robustness and accuracy.

Whilst scaling transformers often leads to better performance on complex tasks, the size introduces problems with gradient updates. The learning rate determines how quickly the model adjusts its weights based on the gradient from the loss function, and it must be chosen carefully in deep networks. Too low of a value can result in slow convergence, whilst too high can give sub-optimal solutions as the model won't be able to find good local minima in the loss landscape. We implement learning rate scheduling, a method to change the learning rate depending on how far the model is into training and the internal dimensionality of the transformer. The learning rate increases at the beginning of training to help it learn quickly, then slowly decreases to avoid getting stuck in sub-optimal local minima. We make it dependent on internal dimensionality as models with larger dimensionality require smaller learning rates to avoid exploding gradients. This process began with a learning rate of $1e^{-5}$, which grew to almost $1e^{-4}$, then slowly decreased it over the rest of the training run.

Careful initialization of the weights of large transformer-based models is important to ensure effective training and convergence. We follow the methodology proposed by [Glorot and Bengio, 2010], named *Xavier initialization*. This method draws the initial weights of a neural network layer from a Gaussian distribution, with zero mean and variance $1/n$, with n being the number of inputs to the layer. This results in the variance of the output of each layer being similar to the variance of the input, resulting in faster convergence.

Dropout is a technique that has gained popularity in recent years, due to its simple ability to prevent neural networks from overfitting [Srivastava et al., 2014]. Given some hyperparameter p, dropout randomly drops $p\%$ of the neurons in every layer it is applied to. This encourages the network to use a larger number of neurons to represent features, increasing robustness. Additionally, since music has a complex structure with many possible continuations of notes and chords, dropout prevents music generation models from relying on a sequence of notes which should make it more creative with the music it generates.

Memory Optimization

Initial analysis showed that using a generic transformer model with no optimizations and REMI encoding would require a GPU, or several combined, with 620GB of VRAM to train on music harmon-

isation. By implementing the optimisations we discuss in this section, we decreased this requirement to 12GB.

To significantly decrease the memory requirement of our model whilst retaining the complexity, we implemented flash attention. At first when working on our implementation, there was no research on using it for cross-attention, i.e., decoder to encoder. Instead, all previous work was done on self-attention, i.e., decoder to decoder and encoder to encoder. A few weeks later PyTorch released their adaptive *scaled dot product attention*[1] which automatically applied the most efficient attention scheme from regular attention, flash attention, and memory-efficient attention[2]. We switched to using their implementation as it significantly sped up all our attention modules, not just in the decoder.

To achieve our optimization goal, we only trained our models on Octuple encodings of sequences that were less than 4000 tokens in length in both sequences, which removed 15% of examples from our dataset. Inference of more than 4000 tokens is possible by only taking the most recent 4000 at a time, rather than the entire sequence. However, our model was still limited by memory to only being able to train on a single batch at a time. Having larger batch sizes is beneficial, as the variance in gradients is reduced, leading to a smoother and more reliable convergence. To get a larger effective batch size we used gradient acclamation; given some hyperparameter n, perform n forward passes and sum the losses, then backpropagate them all together. We found $n = 8$ to be a good balance between generalizability and memory footprint.

Octuple tokenization splits all tokens into eight sub-tokens, which means we have eight values to predict at every timestep. This resulted in eight loss functions, which each independently contributed to the total gradient during backpropagation. When paired with gradient accumulation over eight batches, summing over many terms could give massive gradients, potentially increasing the magnitude of the weights until their values overflow. To prevent this and ensure we have stable training, we clipped the gradient to put an upper bound on the norm of the gradients. Additionally, we added weight decay. Weight decay is a regularization technique in machine learning that penalizes large weights, which prevents overfitting and weight overflow.

Additionally, to increase inference speed to facilitate real-time use of our models, we implemented Bayesian weight compression [Louizos et al., 2017]. This method compressed the weights of a neural network by approximating the posterior distribution of the weights using a smaller set of values, which can reconstruct the original weights with approximation when required. Compressing the weights in such a way leads to lower storage and computational requirements. This does increase the online memory footprint of our model, but our previous optimisations resulted in this being insignificant.

Figure 5.2 shows the difference in GPU memory across different sequence lengths for different models. This was computed by taking several full passes (i.e., a forward and backward pass) with specific sequence lengths and calculating the average GPU memory used after each. We see that our flash attention leads to a linear relationship, allowing our model to deal with long sequence lengths. Whilst Bayesian compression adds a significant computational requirement, it still allows us to use long sequences that are under our memory requirement of 16GB.

[1] https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html
[2] https://github.com/facebookresearch/xformers

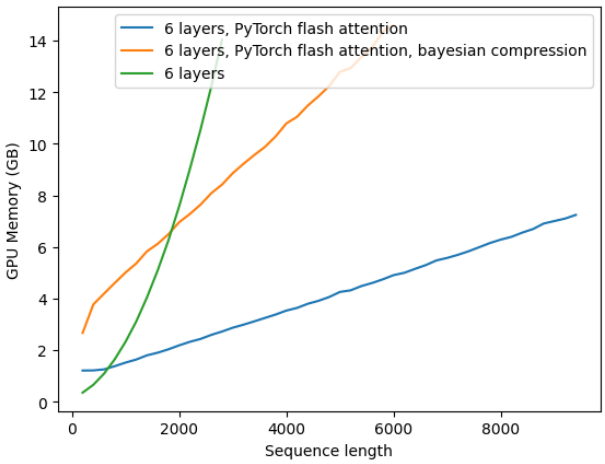


Figure 5.2: Memory requirements of full passes across input sequence lengths

We also explored using lower floating point precision. Floating point numbers are represented by 32 bits, but this has recently been shown to be far more than necessary for storing model weights, with as few as 8 bits being shown to give only a slight decrease in model performance [Wang et al., 2018]. However, when we attempted to use 16-bit precision, we saw an insignificant decrease in memory footprint. This is likely due to the fact that lower precision gives poor speedup for memory bandwidth-bottlenecked models.

5.1.2 MuseFormer

As we have discussed, this task requires a different architecture than the pure generation tasks that have been studied previously. For example, the Museformer model [Yu et al., 2022], while showing impressive generational capabilities would be unable to effectively learn the mappings between our score and performance sequences. We can, however, borrow the attention scheme from that model and implement it within our decoder block.

To implement the attention scheme proposed in the Museformer paper [Yu et al., 2022] we need to accommodate some differences in our model. Firstly our tokenisation method has a key difference in that it doesn't contain a dedicated end of bar token which could be utilised as a 'summarisation' token. Therefore, we added a pre-processing step to insert dummy end-of-bar tokens along with our start-of-sequence and end-of-seqeuence tokens (these are then all removed along with padding tokens when the model decodes a token sequence into MIDI format). Then before the model embeds the token sequences we extract the indexes of the end-of-bar tokens and pass that information to the attention mask creation method.

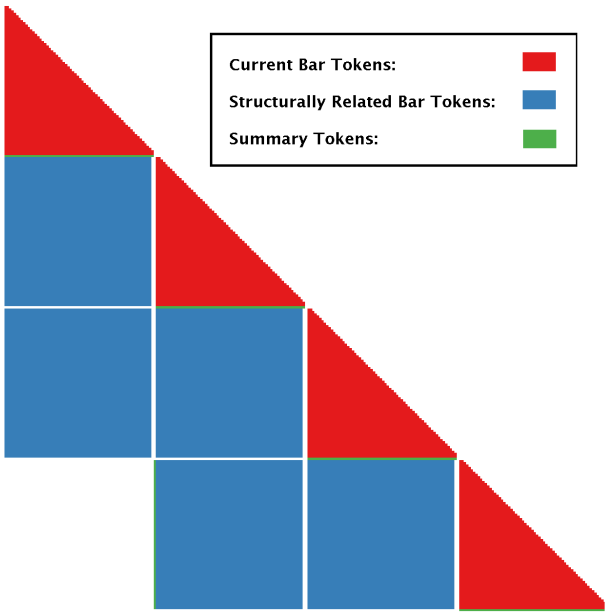


Figure 5.3: An example Museformer attention mask highlighting different reasons for tokens to be included in the mask.

As discussed in section 2.2.3, the Museformer attention scheme makes use of a set of 'structurally related bars' to determine which tokens are included in the mask. The number of bars selected here allows balancing between performance and efficiency, as the more bars that are included, the more tokens there are to which attention is applied, for which more memory is required. In the original paper they propose the 1st, 2nd, 4th, 8th, 12th, 16th, 24th, and 32nd previous bars as structurally related. We also opted to use these bars, as they were designed to be a good general model across a wide range of music. We also experimented with reducing the set to the 1st, 2nd, 4th and 8th bars as a memory optimisation. This reduced the memory complexity of our model further, but after all the previous optimisations discussed this was no longer much of a requirement so we instead opted for the more complex model.

5.2 Generation by Matching Notes

The aforementioned translation-based approach uses tokenization to encode the input, and attempts to produce a sequence of tokens that matches the target. This presents a few challenges that can be difficult to deal with, as discussed thus far. An alternative approach is simply to look at the notes themselves, and transform them directly. This ensures the notes played in both the generated piece and the score contain the same notes, but still allows us to make it sounds human (by modifying the tempo and dynamics). This was a limitation of the aforementioned approach; the newly generated pieces did not always match the piece they were generated from. Additionally, the sequences we have to consider are much simpler.

Whilst we can generate pieces in this way, to compare them to human performances, we need to obtain a mapping from a performance to its score. Indeed, we did something similar when aligning MIDI scores to audio performances (see section 4.2.2). However, the ASAP dataset provides performances in MIDI format as well, which makes this task much easier; we can entirely ignore the Fourier analysis that was necessary in that case. When both files are in MIDI format, we can reduce this problem to that of sequence matching. This is relevant in many fields, particularly in examining DNA sequences. That said, we cannot use exactly the same approach, but we can do something similar (see section 5.2.2).

This is not without its own issues however, particularly in terms of retaining information in previous parts of the piece may be more difficult; if a section is played fast at the start, it should probably be played fast when it is played again later in the piece. Such details may be lost with this expression.

Nevertheless, it is still a promising approach, due to the degree to which it simplifies parts of the problem, the details of which are discussed in this section.

5.2.1 DT Transform

Since we are only looking at the notes, we can reduce each note in the given MIDI file to how it is fundamentally described. This is by 4 dimensions: the time at which it starts, the time at which it ends, the velocity and the pitch. As such, we can convert a MIDI file to a sequence of (start, end, velocity, pitch) tuples to fully describe the piece. From here on, we will refer to this transformation from MIDI to this 4-dimensional space as the DT (dynamics and tempo) transform. Additionally, we implicitly omit the pitch from this transform, as we do not want the model to change this. Instead, we will just use the same pitch as the score.

5.2.2 Mapping

This transform gives us a new problem however, in evaluating the loss of any model implementing a dataset with such sequences. It is easy to compare the start, end and velocity values with the score if these are the only properties changed by the model, but it is not trivial to compare those with the performance. For each note in the score, we would need to know the corresponding note in the performance.

This implies the necessity of a one-to-one mapping from the score to the performance; with this, we can then compare the generated tuple to that which the relevant score tuple is mapped to in the performance. Indeed this is very similar to section 4.2.2, where we used DTW to solve this problem. However, this approach does not necessarily produce a one-to-one mapping, which is needed in this case.

The problem is then to produce a least-cost mapping between two sequences, where each element in the sequence encodes a note as described by the DT transform. We also assert that if a note is mapped to another note, then both notes must have the same pitch.

If the piece was played correctly, this problem would be trivial, as we could simply match up the indices of both sequences. Unfortunately however, in the ASAP dataset, the majority if not all performances have many mistakes: for instance, a missed note, particularly in difficult, high-tempo pieces. Detecting these mistakes is also difficult; in large pieces, many different notes are played, so it is rare that one can single out a note just by looking at its pitch or velocity. Additionally, mistakes can happen anywhere in the piece, so looking at the start and end times in isolation is also ineffective.

Definition of the Recurrence Relation

To compute the mapping, we use a dynamic programming approach, with the following recurrence relation, for two sequences S and T:

$$\begin{aligned}\text{cost}[i][j] = \min(&\text{cost}[i][j-1],\\ &\text{cost}[i-1][j],\\ &|\text{S}[i] - \text{T}[j]| + \text{cost}[i-1][j-1])\end{aligned}$$

This can otherwise be understood as a Bellman optimality equation, that gives the optimal cost for $\text{cost}[i][j]$.

This algorithm is very similar to the Needleman-Wunsch algorithm used to compare DNA sequences. This algorithm has been used extensively in computational biology and bioinformatics to great effect. As such, it works very similarly; for each pair of elements, i and j we can either match them, ignore i (or, equivalently, map it to nothing) or ignore j. This allows us to implicitly identify mistakes,

and remove them, by ignoring the corresponding sequence element, as the cost of mapping it to any element in the score sequence should be high. We allow for this both ways, even though there cannot be any mistakes in the score, because it allows us to exclude any notes in the score that were, mistakenly, never played in the performance.

Normalisation

Naturally, however, we must set a penalty for ignoring elements. If there is none, the algorithm will simply ignore every element in both sequences, as this will have the lowest cost. The appropriate penalty here is not immediately clear; setting it too high will cause it to never ignore a note, and setting it too low will do the converse. In fact, it is not even guaranteed that a constant penalty is sensible in this context, with this expression.

Additionally, we must consider what happens when there is an offset in the performance. Even if it was played perfectly to the score, an offset would cause a difference in the start and end times, and thus there would be some non-zero cost, despite the piece being played perfectly. DTW solved this issue previously but does not produce a one-to-one mapping.

We can solve these issues by normalising both sequences, such that they start at time 0 and end at time 1. Doing so alleviates the offset problem mentioned previously, and is the same approach we used when aligning MIDI scores to audio performances. This retains tempo information; whilst both pieces start at time 0, a note that is faster in the performance will still have a lesser start time than that of the same note in the score.

As every pair of scores and performances are now on the same scale, setting the right penalty is also a simpler problem. However, it is still a hyperparameter, and setting the correct value, as to effectively remove mistakes, is difficult. That said, it is generally better to have a penalty that is too high than one that is too small. It's not particularly problematic if some notes are removed that were played correctly in the performance; the only consequence of this is unnecessarily reducing the size of the sequence, which may make it harder for a model trained on this data to learn. The converse is not true; the consequence of incorrectly including a note for which there is a mistake is high, as this causes the model to learn this mistake. As such, we set it to a high value initially, with the intention of adjusting it as necessary.

5.2.3 Model

There are a few different models we could create for this problem. If we are only interested in making human-sounding music, we can ignore the instructions entirely and create a generative model that takes as input a score and modifies only the note features (not its pitch) for each pitch to make the piece sound like a performance. We can also still consider the instructions, and treat this as a translation task, but we opted to try the former first as it is simpler and will give us a good impression as to whether this representation can give us a good model at all; if it does not work well for the first task, it is unlikely it will work for the second, as it is more difficult.

For this first task, we use a GAN, implementing both the generator and discriminator as recurrent neural networks. The data is still sequential in this representation, and thus a recurrent architecture is still the most appropriate. We started by using an LSTM, as we thought a model that was too complex may overfit, and the representation was fairly simple. The parameters we used initially were a hidden size of 128 and 3 hidden layers.

The main change we made to the typical structure of a GAN for this model is the loss function. Rather than just using the adversarial loss, we also added a reconstruction loss. This ensures that the output is significantly different from its input; though the discriminator should, in theory, be able to distinguish the score from the performance, this may be unrealistic to expect in practice. As such, we can force the generator to output different music by calculating the difference between its output and the score.

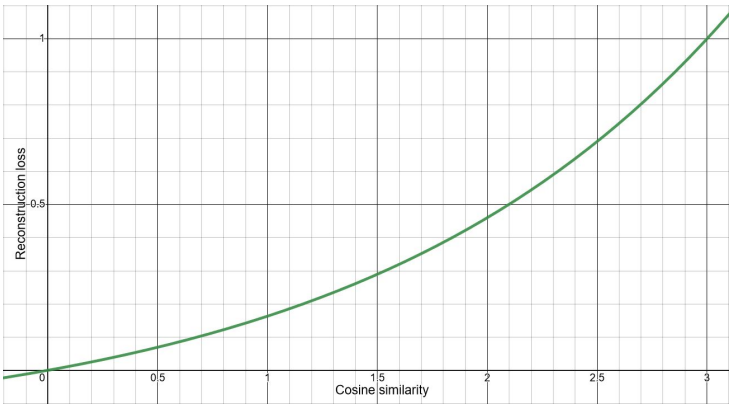


Figure 5.4: Reconstruction loss against cosine similarity.

We do this by examining the cosine similarity between the two pieces, for each dimension, resulting in three values in total. Let the sum of these values be S; then it ranges from $[0, 3]$, as there are 3 dimensions. When S is 0, there is no similarity at all, and 3 if the two pieces are identical. Then, the reconstruction loss is given by $L = \alpha(e^{S/\beta} - 1)$, where $\alpha = \frac{1}{e^{3/\beta}-1} = 0.2$ and $\beta = \frac{3}{ln((a+1)/a)} \approx 1.67$. This gives a function as described in Figure 5.4.

The behaviour of this function is exponential, and can be described by the map $L : [0, 3] \to [0, 1]$. It is defined such that L is 0 when S is 0, and L is 1 when S is 3. As such, we require a low error for S close to 0, and a high error for S close to 3.

There are infinitely many solutions for *alpha* and *beta* that satisfy the above given definitions. These constraints are such that the range of the function is always $[0, 1]$, but the choice of which pair of values, so long as they satisfy the constraints, is somewhat arbitrary, and are chosen such that the function stays close to 0 for $S < 1$, and grows quickly from this point.

We then combine this with the adversarial loss to get the final loss of the model. This is only for the generator however; the discriminator is only updated using the adversarial loss. To ensure the reconstruction loss does not dominate the learning of the model, we decrease its weight as time goes on. We do this because it should learn quickly not to be close to the score, but as it learns to produce better and better performances we won't need to keep reinforcing the fact that it should not be too close to the score, it should naturally exhibit this behaviour. This will allow it to focus more on simply learning to produce better performances to fool the discriminator.

5.3 Generation by Matching Piano Roll

Recall from the previous section, the note-matching music generation approach constructs a note representation of MIDI files by applying the DT transform, and relies on the use of a mapping between two MIDI note sequences. As discovered, finding such mapping can be problematic, because the number note in two MIDI files is likely to differ from each other, thus certain compromises have to be made to ensure a strict one-to-one mapping can be formed.

As an alternative scheme, we adopted a modified version of the matching-based music generation, by comparing piano roll matrices constructed from score and performance MIDI. A piano roll can be seen as a visualisation of MIDI, where x-axis specifies time in the unit of MIDI tick and y-axis in MIDI pitch; the value of each matrix cell is an integer, ranged $[0, 128)$, denoting velocity.

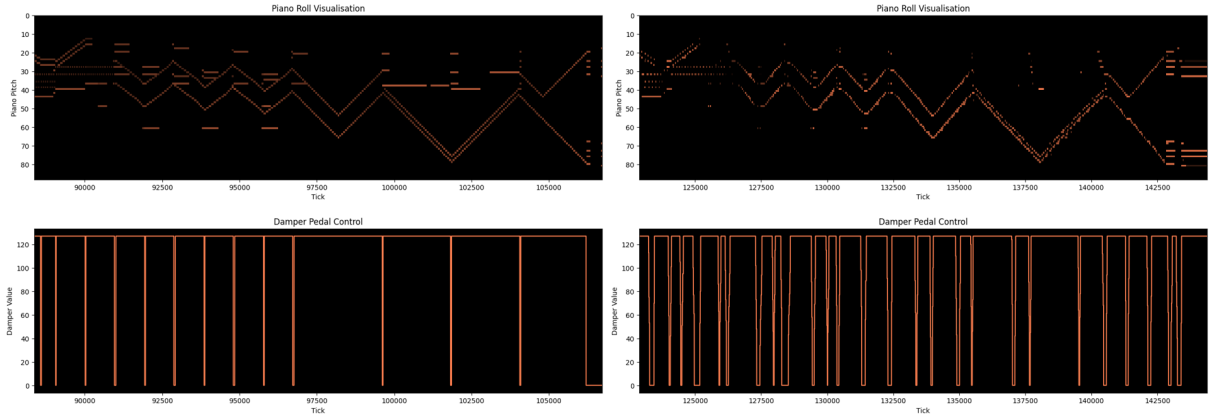


Figure 5.5: Example piano rolls from a *Cadenza* section of Liszt's S.418. Left: robotic MIDI from *classical-piano*. Right: performance MIDI from *Maestro v3.0* dataset.

5.3.1 Model Architecture

The main motivation for comparing two MIDI by piano rolls is based on a number of past research on image, video and audio analysis. As already mentioned in the previous models, the transformer architecture [Vaswani et al., 2017] has gained increasing popularity in these areas because of its versatility and robustness in analysing position-and-time-sensitive data. Although it was originally invented for NLP application, it was adapted later for image classification combining with CNN [Dosovitskiy et al., 2021], and shortly being transferred for audio spectrogram analysis [Gong et al., 2021].

The piano roll can be seen as a special form of image or audio spectrogram, such that the problem of comparing two MIDIs can be reduced to an image/audio analysis problem. To compare two MIDIs in piano roll representation, we planned to design the piano roll comparator model based on vision transformer [Dosovitskiy et al., 2021], a transformer-based image classifier.

However, there is one underlying challenge with the piano roll representation; that is, how the output piano roll should be compared from the model with the performance. Comparison is a vital step to evaluate the loss of a model and enable training. Initially, the dynamic time warping (DTW) algorithm [Raffel and Ellis, 2016] used in reverse engineering tempo (section 4.2) and dynamics (section 4.3) from audio, was considered; this plan was abandoned due to the earlier discovery of inaccuracy of DTW. Thus, it is decided to use a trained model for comparing piano rolls.

DCGAN [Radford et al., 2016] is an image generation model based on GAN and deep CNN. The discriminator architecture is extracted for our use in judging the quality of piano roll output from the transformer. The discriminator network outputs a single unsigned normalised decimal digit (i.e., with range $[0.0, 1.0]$), for which 0.0 indicates fake while 1.0 for real.

Overview

We now present the matching-based music generation model based on piano roll representation. The model is divided into two parts, a generator made with a transformer, and a discriminator with a deep CNN. The transformer will take both robotic and performance MIDI piano roll as inputs; this allows the output piano roll to be constrained to only learning certain musical properties, such as but not limited to tempo and dynamic, so the model is not composing a new piece of music. The output piano roll, presumably humanised, is then fed into the discriminator for judgement.

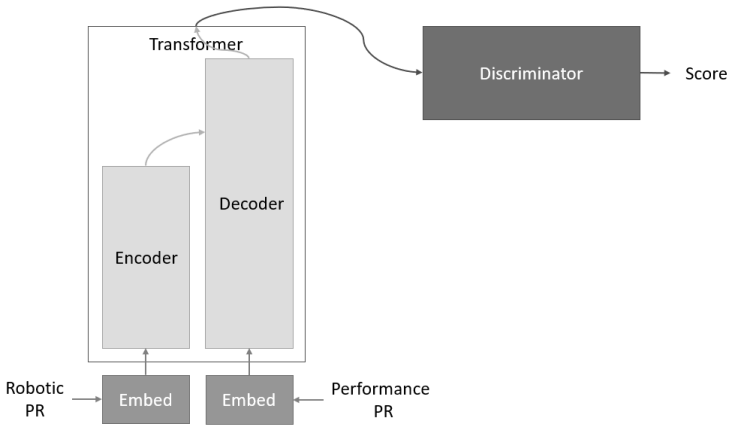


Figure 5.6: The overall architecture of the piano roll matching-based music generator model. Internal structures of the transformer and discriminator are not shown, which will be discussed in detail later.

5.3.2 PR Transform

The input MIDI data needs to be processed and structured into a piano roll before it can be taken by the model. This is achieved through PR (piano roll) transform. To simplify the algorithm, the PR transform is built based on the DT transform as described in section 5.2.1.

The input of PR transform will be a 4D matrix of MIDI note representation after applying DT transform. The dimension of a piano roll is a 2D matrix of size $T \times N$, where T denotes the number of time steps in ticks and N denotes the number of note features. Although the upper limit of MIDI notes is 128, as the project is focusing on piano pieces, N can be reduced to 88 for saving memory. The total time step in tick, T, is determined by the end time of the last MIDI note. The piano roll can then be constructed by filling the matrix with velocities based on the start/end time and pitch of each note.

Control Changes

The piano roll shows its merit of being able to record note properties more than just time, pitch and velocity as in note representation. MIDI control changes can also be recorded into the piano roll, such that the model can learn to adjust MIDI controller as well. There are a large variety of MIDI controllers, but the most significant ones for piano pieces are the pedal controls. Despite this, there are four types of piano pedals. The sustain pedal (a.k.a., damper pedal) is the most commonly used one, and thus it is sufficient to record this type of pedal only to drastically improve the quality of humanisation.

The value range of MIDI control changes is the same as that of velocity, being $x \in [0, 128) \wedge x \in \mathbb{Z}$, so the control changes can be directly fused into the piano roll along the axis that stores 88 notes. Finally, the number of note features N on our piano roll representation becomes 89, with 88 pitches and 1 damper pedal.

Indicator

One commonly used optimisation for training neural networks is stochastic gradient descent. This algorithm utilises the gradient descent method for finding the global minimum of data to minimise loss, but instead of calculating gradients from all training data, which can be impossible if it is too large to fit the entire dataset into the memory, but taking a random subset of the dataset, and this subset is typically referred to as a batch.

To enable piano roll for batched training, several piano roll matrices need to be joined together to form a 3D matrix of size $B \times T \times N$, where B is the size of the batch and it is a hyperparameter to be

tuned manually. Although the dimension N is identical across all piano rolls, T can be variable, therefore the time axis needs to be padded to force each piano roll in a batch to have the same size.

In addition, as already mentioned, the transformer is a time-sensitive model. Thus, certain indicators need to be inserted into the input, to tell the start and end time of each piano roll in a batch. A similar approach as previously used in the token-based music generation task is adopted, by appending special indicator values at the start and end of each piano roll.

Hence, there are three types of indicator to be inserted into piano roll, namely **SOS** (**S**tart **O**f **S**equence), **EOS** (**E**nd **O**f **S**equence) and **PAD**. To avoid duplication with velocities and control changes, the integer value of these indicators are assigned as 128, 129 and 130, respectively. In the end, the numeric range of the piano roll is $[0, 131)$.

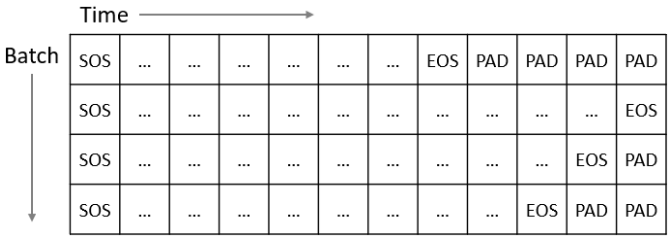


Figure 5.7: An example of the insertion of indicators into a batched piano roll. '...' indicates the actual value of velocity and control changes. The note feature axis N is not shown.

Time Window

Similar to many time-dependent models, transformers also suffer from the limitation of having an upper bounded input length. As an example, the latest GPT-4 can handle a maximum of 2^{15} tokens[1]. Although this quantity is sufficient for most language tasks, it is considerably less than the total number of ticks a MIDI file has. The example performance MIDI file shown in Figure 5.5 has a playback time of 996 seconds, or equivalently, over 760k ticks.

As a solution, it is essential to reduce the input sequence length by reducing the time resolution. This approach was used by MuseGAN [Dong et al., 2018], which uses a piano roll representation with a time resolution of 96 ticks per bar; assuming the time signature is 4/4, this is equivalent to 24 PPQ. We notice that this time resolution appears to be viable for generated music with simple melodies and musical structure, but might be insufficient for complex human-performed pieces, as can be demonstrated in Figure 5.5 showing a large number of short notes being played rapidly.

Hence, another approach of grouping some time steps into a time window is considered, to reduce the length of time axis. This approach is widely adapted in the aforementioned image and audio transformer [Dosovitskiy et al., 2021, Gong et al., 2021]. There are many ways of grouping time steps, but they can be generally classified into four categories:

- ◇ Non-overlapping 1D window
- ◇ Overlapping 1D window
- ◇ Non-overlapping 2D window
- ◇ Overlapping 2D window

For image application, it is typical to use a 2D window to allow the model to learn spatial features. For audio, both 1D and 2D windows are proven to be applicable. Specifically for a 1D window, it is a

[1] https://platform.openai.com/docs/models/gpt-4

better practice to apply the window on time rather than the pitch axis. Regarding the choice of window stride to decide if windows should overlap with each other, past research shows that the overlapping approach achieves a marginally better accuracy [Zhang et al., 2022]. Considering the fact that the non-overlapping window diminishes the length of time more than the overlapping version, it is sensible to sacrifice the accuracy slightly.

It is worth mentioning that the original method presented in those models divides the input data into a number of windows, and feeds each window separately into the transformer. We evaluate that this approach is no longer suitable for music humanisation. First, the transformer is unable to infer any data beyond the current window. Moreover, the input length of robotic and performance MIDI piano roll can be vastly different, hence there is no guarantee that the sequence of notes from the same time windows of two MIDIs of the same piece will match each other.

Therefore, instead of feeding the windows to the model independently, an additional model is added to serve as an auto-encoder. This model is used to encode time steps specified in ticks within a time window into a small vector of embedded features. After executing the auto-encoder, the size of the input piano roll will become a 4D matrix of size $B \times W \times N \times E$, where W is the number of time windows and E is the size of the embedded vector; W should be much smaller than T to achieve an effective dimensionality reduction of time, and $W = \frac{T}{W_s}$ where W_s is the size of time window, i.e., the number of time step in a time window.

In this case, the transformer can consume the sequence of encoded time windows at once, such that the indicator scheme needs to be updated to work with this new adoption. This is done by filling up the entire time window, rather than just one time step, with indicators. Note that if the original number of time steps is not a multiple of the time window size, zero will be used to fill this up, followed by **EOS** and optionally **PAD**. The main intuition of using zero rather than **EOS** directly is to ensure the model can see the full information in one time window, rather than being chopped up into two.

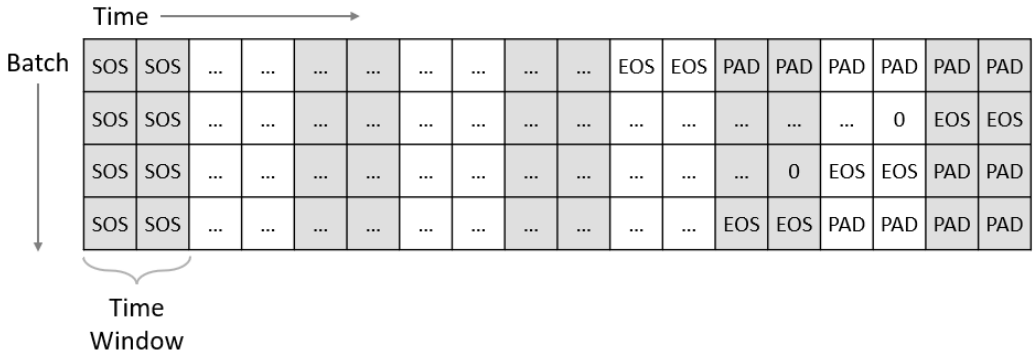


Figure 5.8: The updated indicator scheme that works with a time window, similar to the example in 5.7. The time window illustrated here has a size of 2.

5.3.3 Model Implementation

Transformer

The main adjustment to the embedding layer is the addition of time windowing. The entire input first goes into a regular embedding layer to encode each scalar input in the range $[0, 131)$ to an embedded vector, followed by a deep convolution network for extracting the feature matrix of each time window into a latent space vector. The 4D matrix will then be fed into the transformer. The output network of the transformer consists of another deep transposed convolution network for undoing the time window feature extraction, and for converting the time window back to each time step.

Attention and Padding Mask

The transformer relies on the mechanism of attention to learn a series of data. The attention function requires the use of two types of masks to control the attention. An attention mask is used to prevent the model from peeking ahead of the current location, whereas the padding mask tells the network to ignore certain locations.

The attention mask is generated based on the original transformer implementation without much modification [Vaswani et al., 2017]. The padding mask is amended so it works with our indicator scheme. As previously discussed, indicator **PAD** will occupy a full time window; this simplifies padding mask generation. The padding mask is a 2D Boolean matrix of size $B \times W$, and the cell is set to true if the corresponding time window in the original input piano roll is filled with **PAD**, or false otherwise.

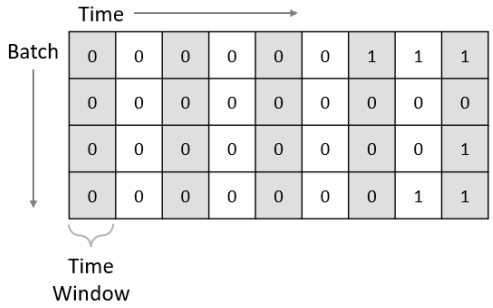


Figure 5.9: The padding mask for the example shown in Figure 5.8. 1 denotes true and 0 for false.

Discriminator

The input to the discriminator has the same size as the output from the transformer, i.e., $(B \times T \times N)$. The discriminator contains a deep convolution network to summarise the input piano roll into a vector of length $(B \times W)$, which is then passed into an LSTM network to summarise features from all time windows into a latent space vector $(B \times F)$, where F is the size of the feature and $(F << W)$. Finally, this is concluded by a linear layer to project the latent vector into a single quality score for each batch, with size B.

Evaluation

Before discussing the individual systems, it is important to recognise that, due to the lack of the necessary datasets, some systems can not be objectively tested. Instead, where necessary, we look at a few samples and measure whether the results align with intuition; if they do not, we ask if they can be justified.

When looking at our machine learning models, a similar problem applies, because there is no ground truth to measure against. Instead, we study the loss functions, look for convergence and see if the results follow intuition.

6.1 Optical Music Recognition

We trained our model on the dataset with 10,000 examples for 8 hours on an NVIDIA T4 GPU. Shown in Figure 6.1 is the convergence plot, and a plot displaying the inverse Symbol Error Rate (SER), calculated as $(1 - SER)$, over training. Beyond the displayed training our model began to overfit, so we omit analysis of such. We see we achieve an SER of 72%, which is impressive considering we purposefully used the most challenging dataset we could find, and since we expanded on the generic OMR task to jointly recognise annotations.

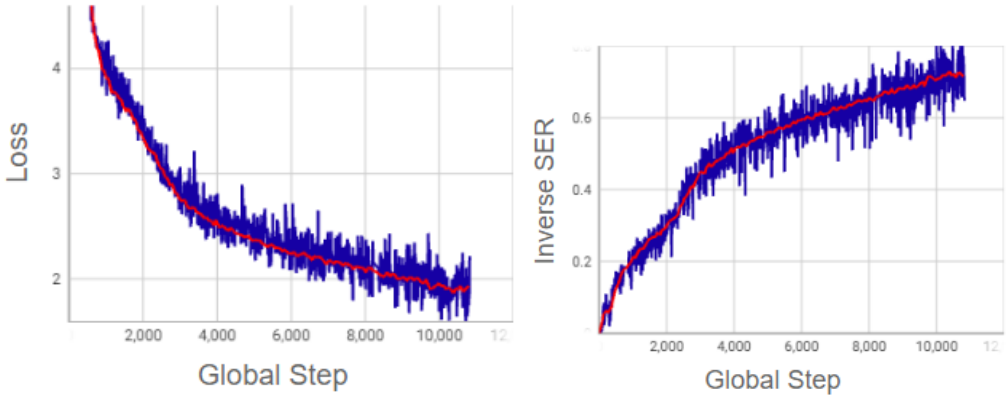


Figure 6.1: Metrics of training our optical music recognition model. Shown is the loss and Symbol Error Rate (SER)

To identify which symbols the model was struggling to identify we created a confusion matrix. Our confusion matrix compares the predicted symbols against the actual symbols, which allows us to see what symbols the model commonly confuses with each other. We built our confusion matrix by testing our final model on the full test dataset, and due to having two separate classification heads we constructed one matrix for each.

Displayed in Figure 6.2 is our confusion matrix for our annotations. Note that each row and column is normalised to sum to one, as there was a significant class imbalance. We can see that the model is

able to distinguish between the short and long annotations well, but often fails to correctly identify the annotation within those two sub-classes. This isn't surprising, as the p and f annotations look somewhat similar on a small resolution. The best performance is for the longer instructions, which is likely because there are fewer of them, so there is less room for confusion in that sub-class.

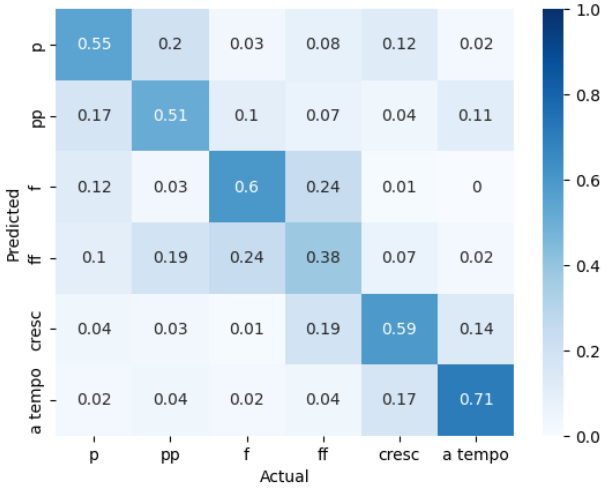


Figure 6.2: The confusion matrix for the prediction of annotations.

6.1.1 Future Work

Our inspection showed that the model struggled to accurately identify annotations. We could solve this by splitting our architecture sooner to allow each of the classification heads more independent processing. This might involve having a smaller transformer decoder, then including more feed-forward layers in each classification head.

We hypothesise that a significant improvement could come from having the model classify longer sequence lengths. Whilst this may seem counterintuitive, as longer sequences are more prone to cascading errors, it would allow models to use contextual information to aid their classification. If the model was conditioned on the entire prior piece of music, it could pick up on frequent repetitions of chords and notes. Note this would significantly improve the computational cost, especially if using an architecture with cross-attention similar to ours.

Finally, we could pre-train our decoder on music generation. Similar to our approach for humanisation, pre-training it on generating music will have it already know musical structures. We could also look into using an off-the-shelf model implemented by others' research, but these are often massive and would likely cause our model to overfit when finetuning. Instead, we could explore *imitation learning*; this is a form of model compression by training a smaller network to copy the outputs of a larger network [Correia-Silva et al., 2018]. Following this, we could compress large state-of-the-art approaches to smaller networks we can use in future OMR solutions.

6.2 Reverse Engineering Tempo

Evaluating the performance of this system is difficult as it is infeasible to try to understand exactly what instructions a human player applied when playing a performance; it is unlikely the human player even knows this themselves, and even less likely after the fact. Ideally we would have a dataset specifically curated for this task, in which the player deliberately plays the piece in a way that is agreed on before they start, and the data is labelled accordingly, but such a dataset does not exist, and is very difficult to create (primarily because a skilled human player is needed to create a performance, of which we need

many). With the resources we have, it is not possible for us to create this dataset, and we cannot create it synthetically either, as the performances would no longer be human.

Instead, we perform black box testing to ensure the results are as expected, at least in form. In addition, we test the system against a few samples and study whether the results align with intuition.

6.2.1 Results

The tests, and their results, can be seen in Figure 6.3. For this test, we compute the difference statistic as the percentage of points where there is a difference across the mapping between the score and performance.

Test ID	Test	Description	Type	Expected	Actual
1	Score - Bach/Fugue/bwv_846/midi_score.mid Performance - Synthesis of Bach/Fugue/bwv_846/midi_score.mid	As the audio file is a synthesis of the score, there should be no changes found.	Negative testing.	`difference = 0`	0
2	Score - Bach/Prelude/bwv_848.mid Performance - Bach/Prelude/bwv_848/MiyashitaM01M.wav	The performance corresponds to the same piece as the score, so some changes should be found.	Functional testing.	0.3 < `difference < 0.5`	0.312
3	Score - Beethoven/Piano Sonatas/3-1/midi_score.mid Performance - Rachimaninoff/Preludes_op_23/Nikiforov14M.wav	The performance does not correspond to the same piece as the score, so a large number of changes should be found.	Negative testing.	`difference > 0.5`	0.856
4	Score - Mozart/Fantasie_475/midi_score.mid Performance - Mozart/Fantasie_475/Huangci05M.wav	The size of the piece is very large, so this test ensures the system works correctly even for larger pieces. The performance corresponds to the same piece as the score so some changes should be found.	Boundary testing.	0.3 < `difference < 0.5`	0.466

Figure 6.3: Test results for the reverse engineering tempo system.

As shown by the results, the tests indeed fit in line with expectation and the system appears to be performing correctly.

However, manually checking the results is also important, as mentioned at the beginning in section 6.2. Here the results we obtained are quite strange, because the actual differences themselves are very small to be noticeable. On inspection, this seems to suggest that the results will not generalise, but repeating the above tests for extra score/performance pairs shows the same results.

6.2.2 Limitations

A possible explanation for this behaviour is the approach itself. Since DTW attempts to compute a least-cost mapping, it is not incentivised to map between notes in the performance and score, because the algorithm has no knowledge of where these are. It simply finds similar points in both pieces, which may not actually correspond to the notes themselves.

Additionally, this may be due to noise in the performance. The audio is taken from live performances, from which there is a small amount of noise in the background, which may be interfering with the results. We would like the system to handle these cases effectively however, since most performances will be like this and it will make the system more robust. Certain adjustments or modifications may be necessary to handle these cases more effectively.

6.2.3 Future Work

For an individual or group in the future with the necessary resources, a dataset to test this system effectively would be very beneficial. Such a dataset should be composed of scores and accompanying performances, as well as labels in the form of the exact tempo changes made. We can then use this

to accurately and objectively measure the quality of the output, rather than heuristically. Naturally however producing such a dataset is difficult, so we were not able to do this ourselves; a skilled musician is needed, as well as the time to go produce a number of performances by hand.

Additionally, the results may be a consequence of the fact that the problem is simply algorithmically difficult to solve. In such a case it would be interesting to explore how effective machine learning is for this problem; this requires a dataset however, which stresses the importance of the first point.

6.3 Reverse Engineering Dynamics

Similarly to the system for reverse engineering tempo, testing this system requires a special dataset that is not available and is infeasible for us to create within the limited project time. Therefore, as before, we perform black box testing to ensure the results are as expected and test the system against a few samples.

6.3.1 Results

We test this system by using the mean absolute error between the dynamics of the input score MIDI file and the target performance audio recording, the result of which can be found in Figure 6.4.

Test ID	Test	Description	Result
1	Score - Beethoven-op81a_mvt3.mid Performance - Beethoven-op81a_mvt3.mp3	A robotic MIDI corresponds to a performance audio recording of the same piece.	14.58
2	Score - Beethoven-op81a_mvt3.mid Performance – Chopin_op58_mvt4.mp3	A robotic MIDI corresponds to a performance audio recording of a different piece.	16.06
3	Score - Beethoven-op81a_mvt3.mid Performance – Synthesis of Beethoven-op81a_mvt3.mid	A robotic MIDI corresponds to an audio recording that is the direct synthesis of the MIDI.	14.86

Figure 6.4: Test results for the reverse engineering dynamics system.

The results demonstrate an increase in difference between MIDI and audio that are not the same piece as expected. However, it fails to show a difference between test cases 1 and 3, where the audio recording is changed from an actual performance audio to a synthesised one.

Similarly, the effectiveness of the reverse engineering dynamics system can also be verified by manually listening to the output. It is noticeable that the dynamics of the output MIDI are changed, but we can argue that such changes do not usually correspond to the volume change in the audio recording correctly.

6.3.2 Limitations

Our speculation for this discrepancy compared to the test in the tempo system is that, even though the time alignment between score and synthesised performance in test case 3 is expected to be indifferent, as can be seen from the test result in 6.3, the conversion from volume in audio to dynamics in MIDI is uncertain, which depends on the choice of velocity curve and MIDI synthesiser.

As the implementation of this system depends on DTW, the accuracy of such is highly bounded by the performance of this algorithm. Consequently, the extracted dynamics in the output MIDI might be mapped incorrectly from the performance audio.

6.3.3 Future Work

The imperfect outcome of this system convinces us that doing signal decomposition based on DTW is not an ideal approach. As highlighted in section 4.3.2, it is possible to achieve a more effective reverse engineering dynamics system by utilising F0-estimation with a trained neural network, provided having access to an abundant amount of dataset for this purpose.

6.4 Sectioning

Evaluating the performance of our sectioning component poses some difficulty due to the subjectivity of what constitutes the correctness of sections of a piece. As such, the main objectives in this evaluation will be to demonstrate that our approach provides results reasonably in line with what a user would expect of the piece. While this testing is not fully comprehensive, it demonstrates the main functionality we hoped to see for this project. The goal of this component being to provide general positional markers throughout a piece which would identify common repeated themes.

6.4.1 Results

We tested a range of pieces by evaluating how closely the sections produced would relate to an initial blind observation of the piece. This allowed us to understand how well a user will be able to use this system to describe the main sections of the piece that they identify with little background information.

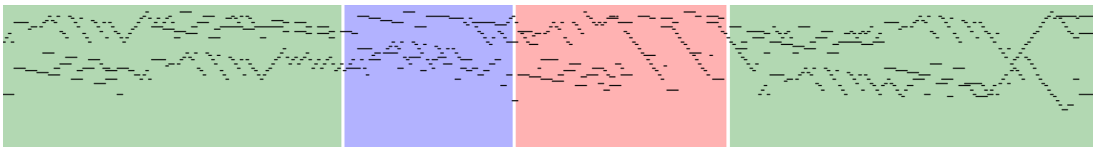
Test ID	Test Input	Description	Outcome
1	Chopin, Sonata 3, 2nd Movement. 2 Sections	This piece has a distinct middle section we expect to be identified and with the start and beginning classified with the same section label. Expect 2 clearly labelled sections.	2 main sections identified; however the end of the middle section is later than we would label by eye.
2	Chopin, Sonata 3, 2nd Movement. 3 Sections	This piece has a distinct middle section we expect to be identified and with the start and beginning classified with the same section label. We expect the extra label to identify slight differences within the one of the two main sections. Expect 3 clearly labelled sections.	2 main sections identified, and a third section breaks up the beginning and end into 2 subsections; however the end of the middle section is later than we would label by eye.
3	Bach, Prelude, BWV-864. 3 Sections	This piece has less distinct sections, expect some common labels between the beginning and end of the piece. Expect 3 clearly labelled sections.	Piece broken into an A,B,C,A structure. Each section describing a relatively even portion of the piece. Beginning and end share labels.
4	Mozart, Turkish March. 2 Sections	The piece is made up of 2-3 repeated phrases with a distinct middle break with a distinct sound to that of the rest of the piece. Expected 2 clearly labelled sections.	Piece broken into an A,B,A,B structure. Each section covers a very general section of the piece. Generally consistent between sections but combines some obvious repeats.
5	Mozart, Turkish March. 4 Sections	The piece is made up of 2-3 repeated phrases with a distinct middle break with a distinct sound to that of the rest of the piece. Expected 4 clearly labelled sections.	Piece broken into an A,B,C,A,B,D structure. Each section covers a very general section of the piece. Models the repeated phrases.

Figure 6.5: Sectioning test results over multiple pieces with different section counts.

What we observed in Figure 6.5 was that most pieces we tried would at least generate clear section breakdowns of the piece and would correctly account for the number of sections given as input. In fact we found this approach could be very sensitive to this input for more complicated pieces, with a higher number of sections being necessary to accurately identify some of the shorter distinctive phrases. This is best highlighted in Mozart's Turkish March, which when given only 2 sections as input is only able to highlight the main distinctive middle section but combines the shorter sections which are clearly identified when given 4 sections as input (as seen in Figure 6.6c and Figure 6.6d respectively). We also generally saw good comparisons between the vague human description of the piece and the generated sections,

suggesting the approach is fit for purpose in allowing users a method of describing general positions within a piece.

(a) Results of test 1.



(b) Results of test 3.

(c) Results of test 4.

(d) Results of test 5.

Figure 6.6: A sample of sectioning results from our testing.

6.4.2 Limitations

While this approach did mostly give us the results we were aiming for, there were drawbacks. Primarily, the number of sections to break the piece up into is a hyperparameter. This means that the user has to provide extra input which will impact the quality of the results which detracts from the ease of use we were targeting with this component of the project. We considered some potential solutions to resolve this, but determined that it was not a priority as it gives the user some control over the resolution of their instructions, and it would remain user-friendly to adjust due to the results being presented visually as seen in Figure 6.6.

Another issue with the approach is the processing time for long pieces. This impacts the user experience by introducing a processing time between the user inputting their MIDI file and being able to see the breakdown of their piece. It may be possible to improve this by reworking the implementation.

Finally, we observed some minor disparity between where a human may label the start of a section and where the model did. This may result in some discrepancy in where instructions are applied and where the user may have intended. These discrepancies are never more than a few notes, so we are able to mitigate these by applying instructions only at a bar-bar resolution instead of individual notes. We are confident in this mitigation approach as the main underlying structure of a piece of music will almost always be separable by bar numbers.

6.4.3 Future Work

No one limitation within this area of the project is debilitating, however the ease of use has room for improvement given some extra development. The main issue we had was processing time as mentioned above. While we have no clear optimisation directions to propose, the current implementation likely has

shortcomings and an exploration into how this could be improved with techniques such as multi-threading could improve the practicality of this component.

Another factor is that the accuracy of this approach while perfectly usable for our purposes, is not faultless. More research into the correct hyperparameters, or finding more modern research into these techniques would be beneficial in producing a more effective system.

6.5 Music Instruction Parsing

As there is no specialised dataset for testing language with music instructions, the test is performed by supplying a number of inputs acting as if they were provided by human users, such that the inputs are formatted in natural language.

6.5.1 Results

Test ID	User Input	Test Purpose	Parser Output
1	Hello world!	A test base case, parser should not give any output on unintelligible input.	▪ Empty
2	Please make section 0 Adagio.	A simple test case that maps a section to an instruction.	▪ Section 5 is played in tempo *adagio*.
3	Now use cresc and stringendo in section 1.	Use a mixture of tempo and dynamic instruction in one sentence. Moreover, the order of specification of instructions and section number should not affect the output.	▪ Section 1 is played in tempo *stringendo* and dynamic *cresc*.
4	I want to play section 4 and 6 in Allegro assai and mp. Then a tempo for section 5. Hmmm, okay. Finally Presto in section 0	Correct handling of multiple sentences, mixture of section numbers and different category of instructions; can successfully ignore uninformative sentences.	▪ Section 4 and 6 are played in tempo *allegro assai* and dynamic *mp*. ▪ Section 5 is played in tempo *a tempo*. ▪ Section 0 is played in tempo *presto*.

Figure 6.7: Test results for the NLP music instruction parser.

The test result demonstrates that the music instruction parser successfully interprets user inputs containing tempo and dynamics instructions to be applied to specified sections. The parser is reasonably robust against faulty user inputs that contain descriptions unrelated to music, as well as recognising inputs with fairly high complexity.

6.5.2 Limitations

The current implementation of the parser utilises a simple search-based approach, and one can argue that the search rules can be insufficient for more relaxed user input. For example, the parser cannot handle effect-over-time instructions such as *accelerando* (gradually speeds up) more precisely by specifying a range of sections (e.g., from section x to y).

6.5.3 Future Work

Given a suitable dataset that contains a collection of sentences with music instructions, this parser can be developed using the statistical approach by training a model, making the aforementioned limitations addressable as a future goal.

6.6 Generation as a Translation Task

This section evaluates the model described in section 5.1. We mostly found success with pretraining on generation, but our models failed to converge when finetuning on humanisation.

6.6.1 Custom Transformer

We started by pretraining on music generation by only using the decoder. Throughout training, we monitored the loss and accuracy of both our training and validation datasets. We also compared the distributions of predicted logits against the ground truth, to manually inspect if any tokens were significantly over/under sampled. This is shown in Figure 6.8; we see that the model converges its predictions to the distribution of the ground truth over time. Note, the ground truth buckets were computed over the entire validation set, whereas the predicted buckets were sampled from individual examples.

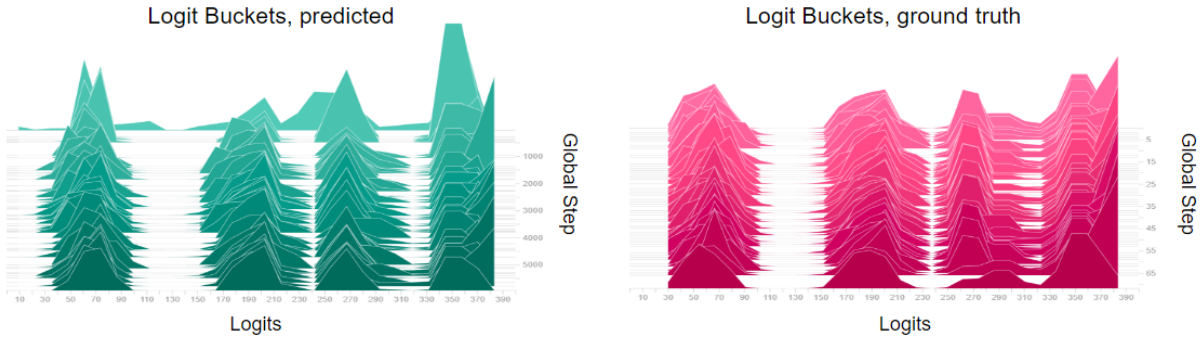


Figure 6.8: Distribution of logits for both predicted and ground truth. The x-axis represents different logit bins, the y-axis shows progression in time during training, and the z-axis (up) shows the frequency of such bin.

We additionally frequently decoded predictions of our model during training on the test set to manually evaluate its performance. Interestingly, we saw the model learn increasingly complex structures as the training progressed. Displayed in Figure 6.9 is a decoded example from early in training. Whilst the model has formed several chords, these are arbitrary and do not follow a key. Additionally, the model has not yet adequately learnt to use rests, as it often predicts several in a row when it could just predict one longer one.

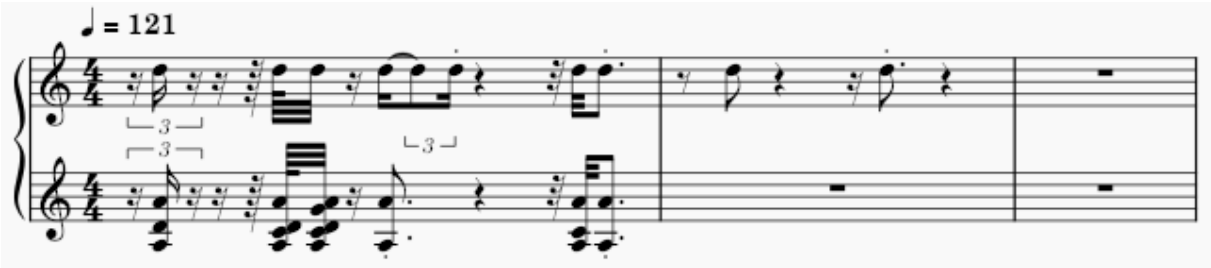


Figure 6.9: Decoded example from custom transformer during early stages of training for music generation

Below in Figure 6.10 are the training metrics from pretraining. The model quickly converges to an accuracy of 47%. Previous research has shown this to be more than sufficient to produce long sequences

of good music; predicting different sequences can still result in a similar sounding performance [Huang et al., 2018].

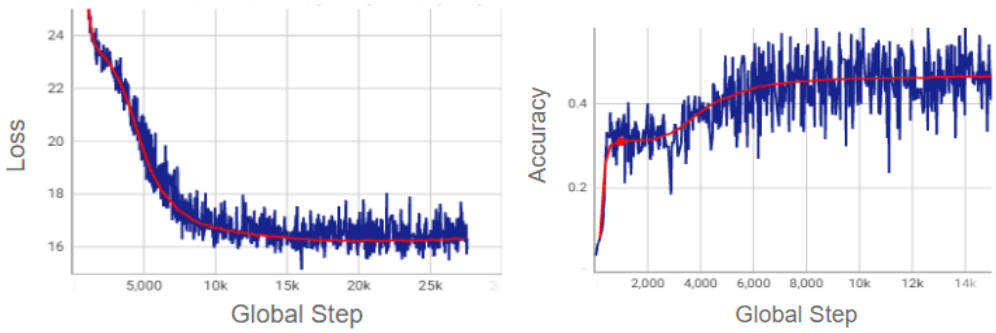


Figure 6.10: Metrics of pretraining our custom transformer on generation

We then finetune our model on the task of humanisation using the ASAP dataset. We unfroze the encoder weights of our architecture, passed the synthetic scores through the encoder, and allowed the decoder to use the cross-attention module to attend to the encoder outputs. Unfortunately, we saw poor convergence. Displayed in 6.11 are the metrics from training our model for 48 hours. The loss quickly decreased, but saw little improvement over the course of training. We also both the loss and accuracy to be very noisy, which was a result of our relatively small batch size of 8.

We theorise this was because of a lack of data, and insufficient complexity of the model. Our model quickly achieved an accuracy of 30%, which was likely due to the pretrained decoder having an understanding of typical structures and continuations in music. However, with the ASAP dataset only containing just over one thousand examples, there was insufficient data to learn the mapping relationship. Additionally, as we noted earlier the ASAP dataset has challenging examples where the score and performance sequences are significantly different, due to tokenization problems with different tempos. Not only did this incur an unnecessary computational burden, but it also presented a difficult relationship to accurately model.

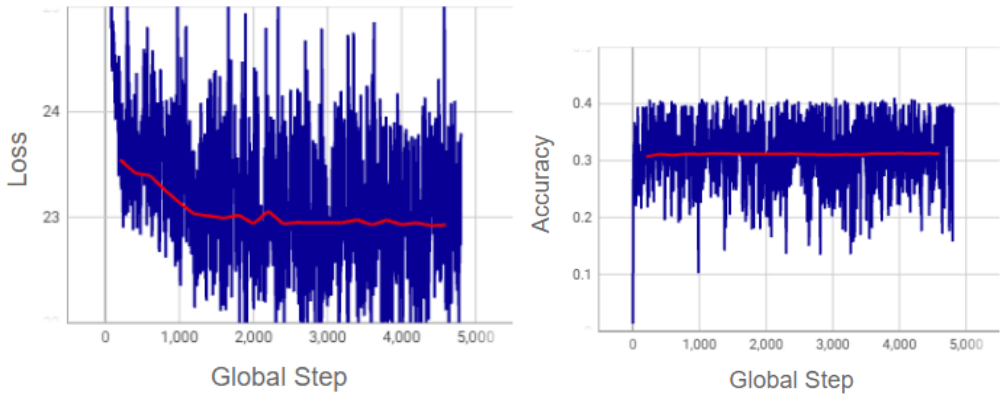
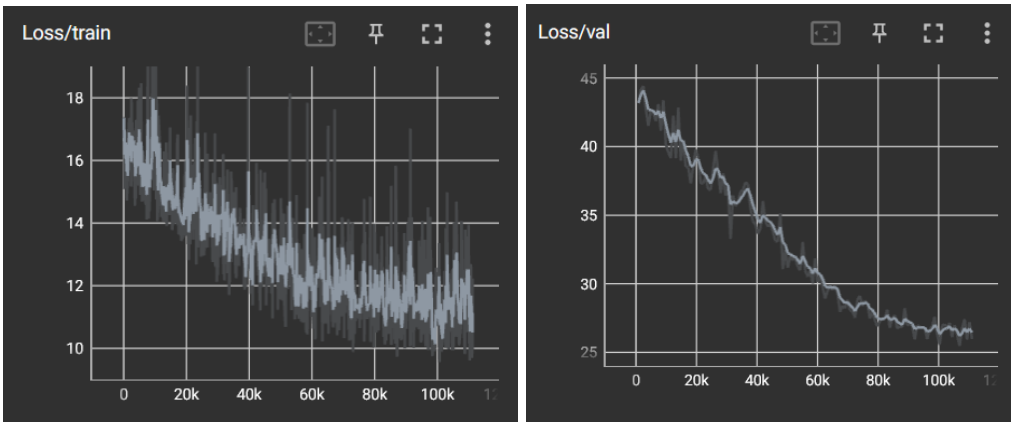


Figure 6.11: Metrics of finetuning our custom transformer on humanisation

6.6.2 Museformer

The introduction of the Museformer attention scheme did not help with this problem much. We did see the model begin to learn more consistently than previously, but the rate of learning began to dramatically slow over time. We also observed a disconnect between the validation and training loss of the model here. However, with the right hyperparameters the model avoided overfitting with the validation loss following the same trends as the training loss.



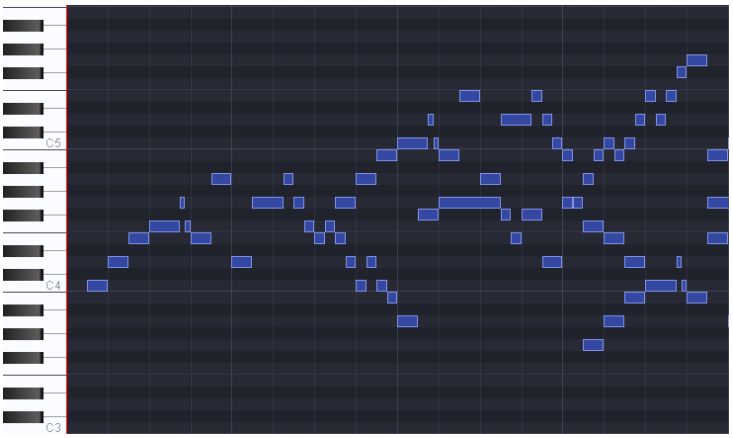
(a) Training loss **(b)** Validation loss

Figure 6.12: The loss of the translation model utilising Museformer Attention

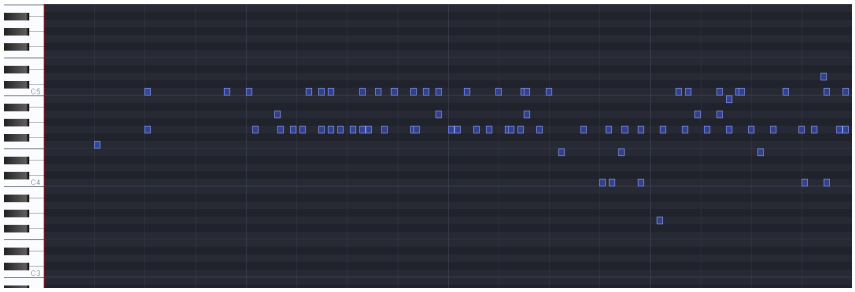
As above, we would intermittently generate a sample piece using data from an unseen dataset. This allowed us to compare the abstract loss value to an actual piece of music and hear the results. Unfortunately, this was never very promising. Even the most 'musical' sounding outputs, were far from the original piece they were intended to mimic, and improvements did not seem particularly tied to the loss function. In our final training cycle, the final model, which showed supposedly the best performance on both the training and validation set, produced the outputs seen in Figure 6.13. As you can see, the output shows almost no resemblance to the original piece, and in fact is mostly comprised of the same two notes being rapidly played. This is a trend we started to observe quite late into the training cycle. We would see large decreases in loss, across both training and validation sets, only to see the outputs filling any empty time with the same rapid playing of notes, drowning out any actual attempt at replicating the piece. We believe this is likely a local minimum in the loss function which the model has been unable to escape.

It is interesting to note that the musical structures formed by the model during pretraining appear to have been lost with the introduction of the new attention scheme. It appears that while this improves the models ability to improve upon its loss, in doing so it is disregarding some of the structurally important aspects it showed promise of in the pretraining process.

Despite the problems described above we still see a slow decrease in the loss function, with no sign of stopping. This gives us confidence that given time the model may still be able to improve from this point given more time and potential tweaks to hyperparameters such as learning rate. This may take a long time however, and is certainly not indicative of a successful model, especially considering the amount of training time used to get to this point.



(a) The source MIDI 'score'.



(b) The generated MIDI 'performance'.

Figure 6.13: A comparison between the first 4 bars of the original piece and the translation model's output (Data sourced from the unseen test set).

Limitations

Unfortunately, as we have seen in our results the performance of our model is not what we would have hoped. We believe there are a few factors that impacted our model's capabilities which we hope can be mitigated with future work to improve our results.

Firstly, as we can see that in our loss functions the models are learning but it is very slow progress. This reinforces our belief that the model does have the potential to eventually learn to complete our task in some capacity but due to underestimates in the complexity of our task and a hard time limit with the deadline of this project we cannot allocate more time to see the full limits of our model. The main issue we are seeing in its current state (which the model may learn to fix eventually) is that the results it generates do not follow the source piece resulting in outputs which are hard to describe as performances of the original piece in any way. This may be a limitation of the model architecture. Currently, the model has access to an encoded state of the whole original piece, but we may see improvements by borrowing ideas from the Museformer paper [Yu et al., 2022] by limiting which tokens the decoder can access when predicting the next note to generate.

It is hard for us to fully explore the limitations of our model architecture itself due to the aforementioned uncertainty about the fully trained performance. However, we can highlight some potential factors for the model's failings in the training so far. One issue we have seen is the model's problems with generating outputs which follow the original piece, which is likely due to the differences between the source and the target sequences being too large and as such the encoder block struggles to effectively pass on useful information to the decoder block for generation. The limitation here may be that the loss function does not incentivise accurate reconstruction enough, or it could be that our encoder architecture is not sufficient to learn the mappings required in our dataset.

6.6.3 Future Work

Due to the nature of our results, much of our proposed future work on this model to is acknowledge the shortcomings in the design and planning of our work in this report. We still believe our models show promise of being able to tackle this problem with extra accommodations considered, and would very much like to see the possibilities of the model given the right environment.

Expanded Dataset

As we have referred to repeatedly through this project, data availability is a substantial problem for this task. We have discussed some alternative approaches we employed to attempt to train our models, but in many ways these are not fully sufficient. As such, for future improvements on this work a reasonable first step would be to improve the availability of this paired data, whether this be manually collected or autonomously generated.

Structurally Related Bar Detection

A potential area of improvement for the Museformer attention scheme is in the structurally related bar choice. The original paper implements the model as a pure generation model so the choices for structurally related bars must be generic and encode an expected structure from a wide range of music. For our task this is not a necessity. As our task is translation, we have the benefit of the source tokens as a reference point for our generation model. By making use of this fact we may be able to design a smarter way of generating the attention mask.

This could involve pre-processing the source piece to find the similarity metrics between every bar and its predecessors and saving the indexes of the top n bars to be used in generating the mask. This may have the benefit of the model being able to more accurately follow the original piece of music as it will have a relative reference of where to look to aid its generation.

Potentially the differences between the source piece and the target performance could make this challenging however. We have seen many pieces in our dataset where the performance varies drastically in tempo to that of the original score. This would mean that while two bars may be similar in the source piece, the time frames which the bars refer to in the target piece may be vastly different, potentially leading to very confusing sounding results.

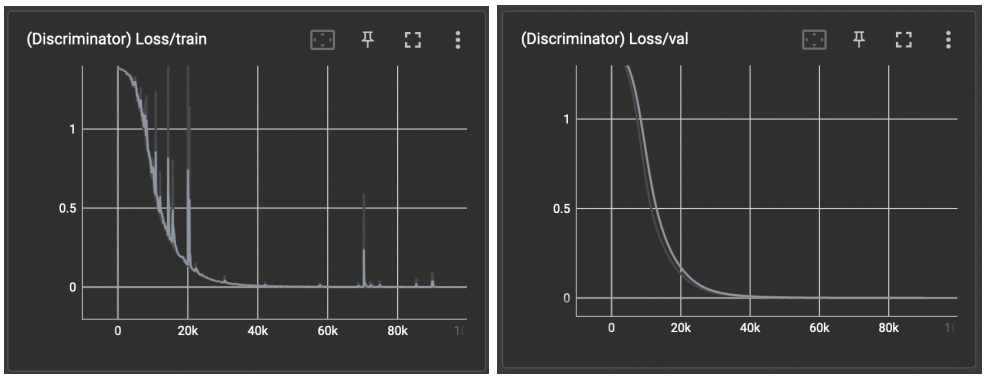
Loss Function

While observing the learning process of our model we saw that reductions in loss were not always a sign of improved generation performance. Specifically, as we mentioned in our results, we would see supposedly large improvements in performance only to be met with a model that repeats a common note rapidly. A potential area of improvement for our model would be to investigate alternative loss functions that may better correspond to our problem, for example finding ways to better represent the reconstruction error of the target sequence.

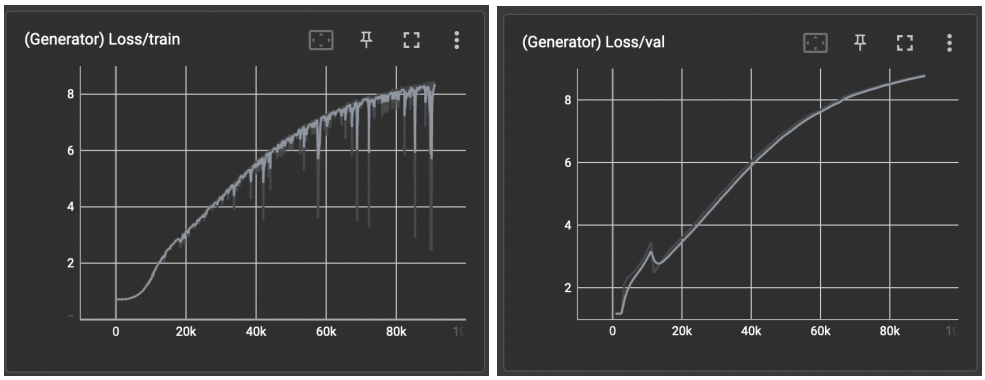
6.7 Generation by Matching Notes

In this section we evaluate the efficacy of the generation model described in section 5.2. These results in this section were produced with the reconstruction loss omitted. To see why, see section 6.7.2.

6.7.1 Results



(a) Discriminator training loss **(b)** Discriminator validation loss



(c) Generator training loss **(d)** Generator validation loss

Figure 6.14: The loss of note matching model during training and validation.

The discriminator learns effectively, and we can see that its loss decreases over time. This is true both for the training set and the validation set, and we can see that it is not overfitting. The generator on the other hand does not show this behaviour. Its loss actually increases, almost monotonically. Whilst we expect some increase in loss in the early stages, it should start to decrease over time.

Indeed, the behaviour we would like to see overall is the loss of the generator and discriminator to approach the same value, since as one goes up the other goes down. What we see here is that only the discriminator seems to be learning.

6.7.2 Limitations

These are not the results we expected, but nonetheless, there are a few reasons that this could be happening. To learn a certain kind of task, a model must be sufficiently complex to accurately model the task. It may be the case that the model we used was not able to capture the task well, due to it being too simple. To remedy this, we experimented by changing the hidden size used by the LSTM. However, this led to very erratic loss functions, with neither model showing any decrease in loss. Whilst it is possible that we were simply unable to find the correct level of complexity, it is much more likely that lack of data is the problem.

Reconstruction loss

In expectation, the reconstruction loss between a given performance and its score should be fairly low; ultimately we are aiming to produce the performance with the model, so if it does not have low loss, the loss function is not sensible. Indeed, this implies the cosine similarity between a performance and its

score should be low, as the function is exponential in cosine similarity. This is not the case. The cosine similarity between a given performance and its score, on average, is very close to 1 in each dimension (start time of notes, end time of notes, velocity of notes).

There are a few possible reasons for this. The first is the approach with which we measure similarity; cosine similarity may not be sensitive enough to find the subtle differences in performances against their scores. We could scale the difference, but we don't know prior by how much; if we scale it such that performances have 0 reconstruction loss on average, this may not be aggressive enough to encourage the model to learn to produce different results, resulting in the same effect as if we removed it entirely.

Another issue may be the matching algorithm. The penalty may be causing too many notes that have deliberately been played differently to be removed. Ultimately the algorithm attempts to produce two sequences that are as similar as possible, which may not be the best approach, as it could cause too many of the characteristic differences in a performance to be lost.

Lastly, the performances may not be diverse or different enough. The difference in reconstruction loss comes from the human player inherently playing the piece differently; if this isn't true to a significant degree, then we shouldn't expect the difference to be high. Unfortunately, we have no reference for the ground truth. It is not in the dataset, and asking the person that played the piece is unfeasible.

Pretraining

Unlike the other models, pretraining for this task is very difficult. We cannot simply pretrain on generic music generation, because the task is too structurally different than creating a new piece from scratch. This, coupled with the lack of data, makes improving this model more challenging.

6.7.3 Future work

In the future, the main work to complete is tuning the various parts of the model.

Matching

We have mentioned previously the importance of having the correct penalty. Finding the correct penalty is a difficult task, and it is difficult to evaluate whether the set penalty is correct. Without fully training the model it is difficult to know if the penalty is suitable, which is very expensive. In addition, there are many other parameters to consider in the performance.

As such, it is more reasonable to approach this problem by finding a strong theoretical justification for a penalty, rather than trying to find it empirically. In the future, we would hope to explore this domain further, and find stronger assumptions we can make to simplify this problem.

Loss

Whilst the reconstruction loss we implemented was not as effective as we hoped, the idea is still sound, and we would expand on it in future work. In addition, we could explore different types of loss that could be more suitable for this problem.

As for the reconstruction loss, there are two directions to explore; measures of similarity, and scaling. The scaling function determines how much to penalise degrees of similarity. For our implementation we use exponential, but it may be worth exploring different growth behaviours. Additionally, aside from cosine similarity, there are other similarity measures that could be explored. Euclidean is usually not preferred for sequence-alignment tasks, but it may be suitable for this task; future study is necessary to understand if this is the case.

Pretraining

Whilst we can't pretrain the generator easily, we can pretrain the discriminator to distinguish between random noise and real performances, which are abundant. This may improve the performance of the generator as well, as it must produce better pieces to fool the discriminator. This is something that can be looked at in the future.

6.8 Generation by Matching Piano Roll

This section evaluates the model described in section 5.3.

6.8.1 Results

In general, it can be seen that while both the transformer generator and discriminator show some learning, the discriminator is learning at a much faster pace than the generator does.

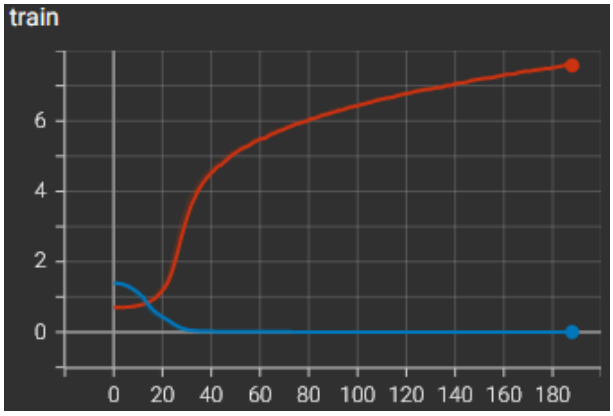


Figure 6.15: Loss during training of the piano roll matching model. Blue: discriminator loss; Red: generator (transformer) loss. Note that the loss of the generator is expected to go up.

6.8.2 Limitations

In this section, the limitations of the model are described.

Memory Consumption

Despite the previously outlined advantages of using the piano roll representation, one significant constraint of the piano roll is its memory efficiency. As can be noticed, even for a complex music piece shown in Figure 5.5, the overall density of music notes is sparse, such that a majority part of the piano roll matrix is empty. However, the piano roll is structured as a dense matrix where time is measured in ticks, resulting in significant memory usage.

This has caused difficulty in training the model, even with our proposal of using time windowing. Particularly, we observe a slow training time due to the required memory being too large.

Reverse PR Transform

Another limitation of the piano roll is its inability of distinguishing whether there is a single note, or multiple notes at the same pitch being played next to each other [Yang et al., 2017], such that the process of converting from piano roll back to note representation, and so a MIDI file, may be lossy and

erroneous. One possible way to mitigate this is by adding an onset marker, as suggested by the original paper, to the beginning of each note.

6.8.3 Future Work

Based on the background study on the generation of music by matching piano roll listed previously, we continue to evaluate that this approach is viable since it is noticeable that the model is learning, despite suffering from a number of practical limitations.

Optimisations need to be taken to reduce memory usage. One possible way is by using gradient checkpoints to prevent caching results from the forward pass, although this can severely impact the runtime. Alternatively, the model can be trained with half-precision floating point, which acts as a trade-off between a slight decrease in training accuracy and a reduction in memory consumption by half. Additionally, we can integrate distributed parallel training to spread the training task across multiple computing nodes.

Project Management

This section documents the way this project was managed, and what motivated our management methodologies. The limitations of these are discussed, and we comment on how we could improve on them in future work.

7.1 Development methodology

Our project followed an agile approach, similar to Kanban. We maintained a backlog of tasks to complete, which was reviewed in weekly meetings. Each meeting, progress on previous tasks was evaluated: if the task had not been completed, any encountered problems were identified, and rectification tasks were added to the backlog. After a discussion with each member, if new problems were identified, tasks to address these were added to the backlog as well. At the end, minutes were taken[1]. This process is shown in Figure 7.1.

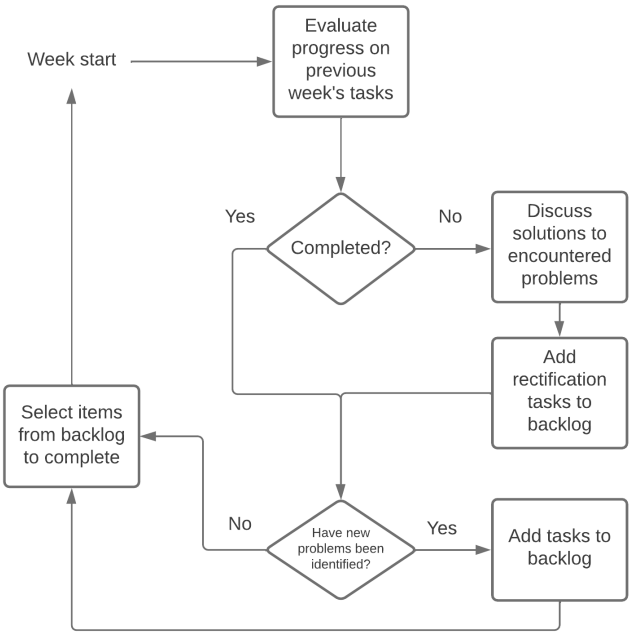


Figure 7.1: Development methodology for this project.

We chose this approach due to the nature of our project. A large part of the project involved exploration, so we expected the need to change previous requirements and add entirely new ones. This

[1] https://docs.google.com/document/d/1Pem_7qMsnPJpPvHbRDzjnbEwc7oAe0P8baY5ze6fyCo/edit?usp=sharing

approach allowed us to stay flexible in the way we approach problems, whilst still giving us a structured approach to tasks. It also allowed us to easily move between tasks as necessary, and proceed quickly from one to the next as they were completed.

To manage and track progress across each member, we selected Aniket to be the project manager. For the project manager, we delegated the following tasks:

- Organising meetings - identifying topics that required discussion each week, ensuring each member had a chance to talk and discuss any of their issues or ideas;
- Minutes - recording what was discussed in each meeting for future reference;
- Communication - liaising between the group and the supervisor, and keeping track of each member's progress during the week.

In addition, we organised two meetings per term with our project supervisor. In these meetings, we consolidated and summarised work thus far, and discussed what was still necessary for the rest of the term. We also used the time to raise any specific issues that had been encountered.

7.2 Schedule & timeline

The first few weeks were spent finding the right direction for our project. We were certain we wanted to focus on music, but initially we attempted to create an OMR system that read sheet music, interpreted the instructions, translated it into MIDI format and performed humanisation. By week 3, following many group discussions and conversations with our supervisor, we narrowed the task down to what it is now: humanisation while following user given instructions.

We then completed our project specification. In doing so, we designed a schedule for term 1 and term 2. These were Gantt charts, showing what work would be completed each week, how long each task would take and when it is expected that it would be finished. Doing so ensured that we had enough time for each task, and allowed us to track our progress thus far. These are given in Figure 7.2 and Figure 7.3.

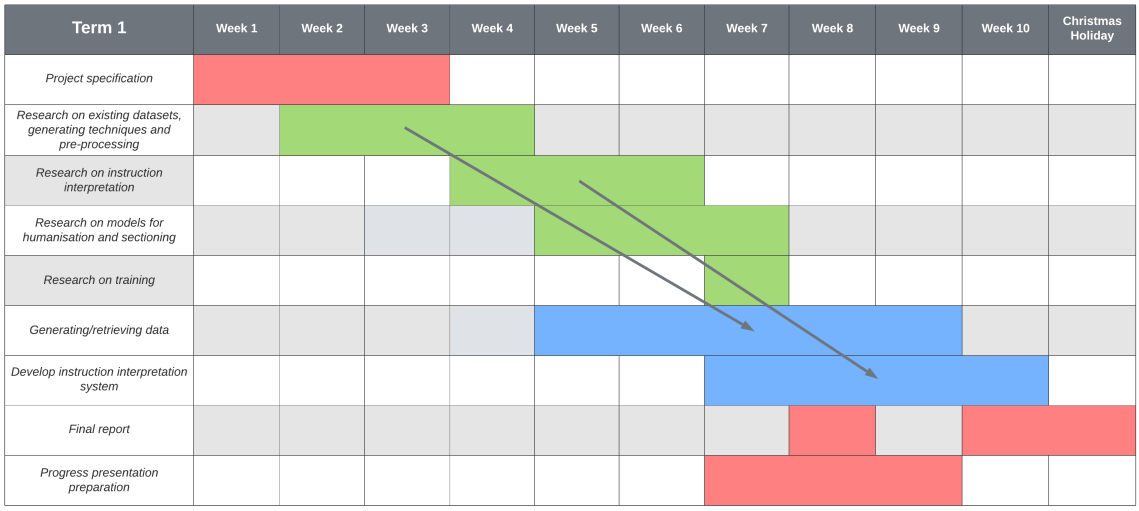


Figure 7.2: Our prospective schedule for term 1, from our project specification.

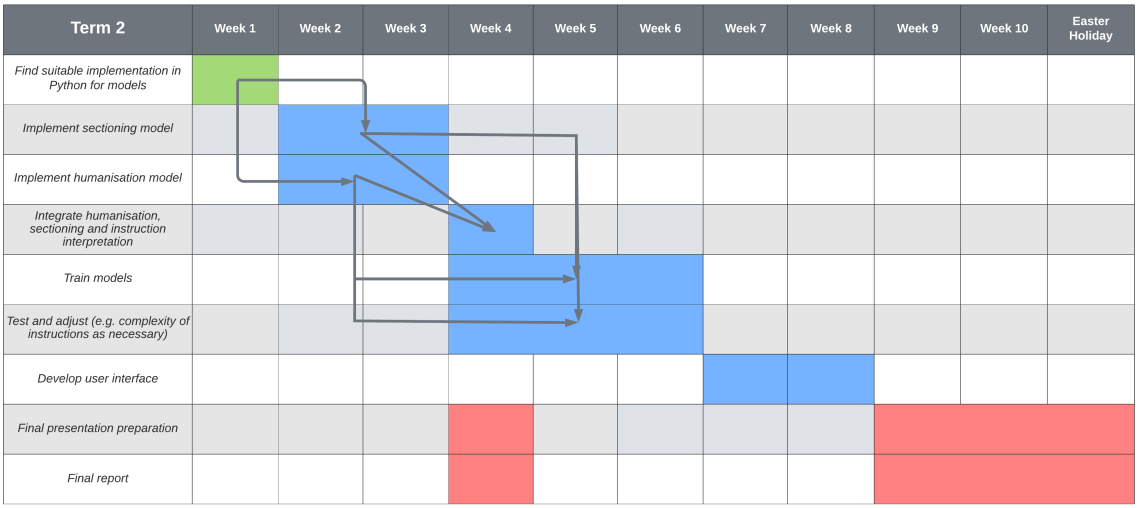


Figure 7.3: Our prospective schedule for term 2, from our project specification.

The arrows in the figures depict dependency; the task at the beginning of the arrow must be completed before the task at the end. For some tasks there is overlap between the connected tasks, which describes tasks for which the next task can begin once the previous task is in its final stages.

Ultimately, these were not reflective of the work done; the primary reason for this was our lack of understanding in parallelising tasks. Much of the work was able to spread across multiple members and completed in parallel, putting us well ahead of schedule for term 1.

More specifically, in term 1, we were able to find a dataset, complete the sectioning model and both instruction interpretation models (OMR and reverse engineering), when initially we had only planned to complete the OMR model. This meant redesigning our term 2 schedule: this was done in preparation for our progress presentation, given in Figure 7.4.

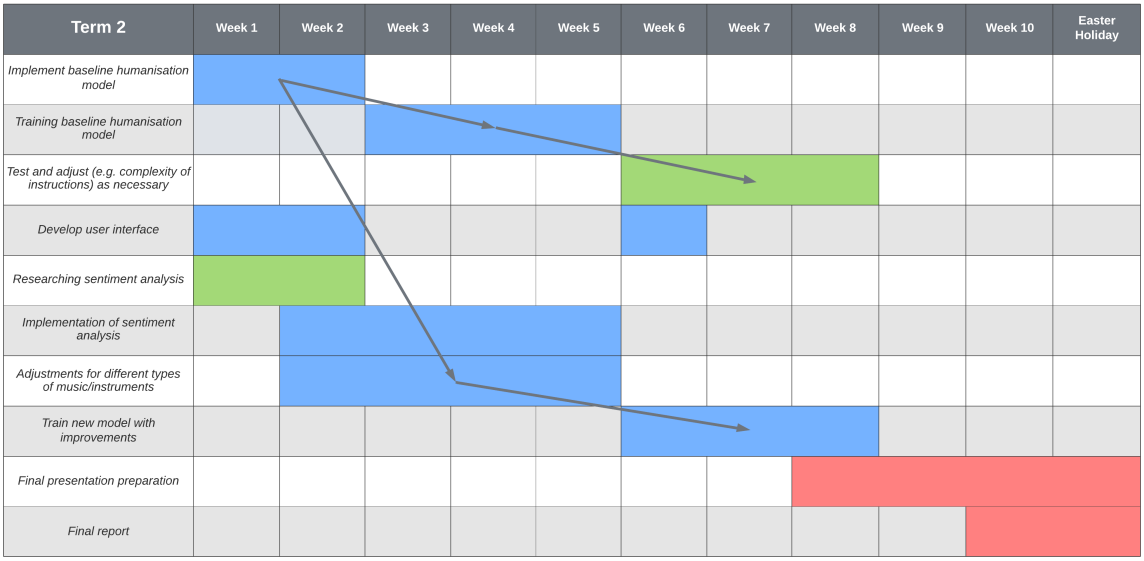


Figure 7.4: The second iteration of the term 2 schedule.

With the start of term 2, due to other responsibilities, we were only able to resume work on the project from week 3. We began by creating a dataset. Whilst we had found a dataset, we needed to process it such that it could be used for our machine learning models. This took longer than expected, due to a large number of edge cases that we had not expected. Additionally, we mistakenly assumed that we would need to know the exact format (after it had been processed) to start implementing our models, but this was not the case, as PyTorch, the framework we used, was very flexible in this regard. This meant the development of these machine learning models was somewhat delayed as well.

Originally we had planned to implement existing models, with some small adaptations. Unfortunately this was not possible, as most of the models we found either weren't applicable to our problem (either because they required a very specific format for the data, or because they solved a very niche problem that was difficult to generalise). As such, except for one, the models had to be implemented from scratch. The exception was a model called Museformer, which still required a significant degree of adapting.

We still finished by the end of the term however, and moved to training. Since we were working with neural networks, significant computational resources were needed. We were not able to freely access these resources as they were in use by others, which caused some additional delays, but we were still able to complete the project in time.

We would have liked to test the models informally by surveying people and consulting them for feedback we could use to improve the models. Due to the time constraints this was not possible, but is certainly something we would look to for future work.

7.3 Tools

We used a number of technologies and services to best support the process we followed to complete our project.

7.3.1 GitHub

GitHub is a hosting service that implements the version-control software Git. Using it allows an individual or group to host source code on the platform, maintain the correct versions across different systems and users, recover lost work and safely make changes to existing software.

We used GitHub to host our project. To do so, we used one main repository to host the project's work, and used different repositories for different sections, which were combined at the end into the main one. Doing so helped us to work better collaboratively, and ensured consistency across the different parts of the system. Splitting the project into repositories, rather than branches, was ultimately done due to scale; for entirely different problems, we thought it best to divide them more definitively, so as to not introduce unwanted behaviour as an artefact of an unrelated part of the system. Indeed, this made unit testing much simpler, and did not affect integration testing significantly.

7.3.2 Overleaf

Overleaf is a collaborative LaTeXeditor. Being able to typeset documents collaboratively has helped us complete documentation quickly, whilst being able to revert changes when necessary. It has also helped us better maintain our project, by organising the structure of documentation and the relevant figures and references.

7.3.3 Discord

We use Discord, a messaging platform, to communicate, and keep track of any discussions. This includes details about the project, as well as minutes for meetings. It's also allowed us to collaborate effectively without everyone being available, such as during the break between terms 1 and 2.

Rather than use an additional service, such as Jira, to maintain our backlog, we decided to use Discord for that as well. Whilst we did try to use Jira at first, it was much more convenient for everyone to use Discord as we were used to checking it frequently to communicate, and the extra functionality from Jira was not needed.

7.4 Risk assessment

The risk matrix for this project is given in Figure 7.5.

Risk ID	Risk	Severity	Occurrence	Detectability	RPN (Risk Priority Number)	Action
1	Suitable dataset cannot be found	9	5	1	45	Avoid
2	Dataset cannot be prepared, cleaned or transformed as necessary	5	2	2	30	Mititgate
3	Research is insufficient regarding sectioning, preprocessing, music information retrieval, music generation	4	4	1	16	Accept
4	Illness, sudden responsibility or other circumstances that impact a team member	6	3	1	18	Accept
5	Loss of work or data	9	2	2	36	Avoid
6	Abandonment of external libraries or resources	6	1	1	6	Accept
7	Insufficient computation resources (memory, GPU processing power, or other)	8	5	1	40	Mitigate

Figure 7.5: A matrix describing the identified risks for this project.

This amends some parts of the risk matrix from our specification, as well as some new risks that were not considered previously. Below we detail for each risk what could cause it, and how it should be dealt with.

Risk 1

A suitable dataset may not be found due to a lack of real-world examples from which a dataset can be constructed, or the availability of such a dataset (for instance, it may require authorisation that we do not have).

Without a suitable dataset we cannot train any machine learning models, and thus our task cannot be completed. Therefore, the severity of this problem is very high. The occurrence is difficult to estimate, but based on the fact that generating such a dataset ourselves is difficult, we expect that it is reasonably possible. However, datasets are incredibly important and continue to grow in necessity. Music generation is a popular machine learning task, and so we estimate that there is a good chance one exists. If a suitable dataset does not exist, we will be clearly able to detect it, as it is a, not literally but conceptually, tangible object.

As such, the contingency plan for this risk is to avoid it. To do so, we used a number of preprocessing techniques to aid in the development of such a dataset, described in chapter 4. In the event that, even

then, we cannot find a dataset, we would need to find a new problem to consider as the project would no longer be feasible.

Risk 2

It may not be able to make the necessary changes to a dataset due to the way it is licensed, or a lack of the necessary tools.

This indeed poses a problem, but it does not threaten the feasibility of the project. The exact effect depends on what exactly we cannot change that is deemed necessary, but nonetheless we would still be able to produce the promised software, albeit with less functionality or quality than was expected. Hence, the severity is 5. The occurrence of this happening is quite low, as datasets often need to be changed in some way to be used, so it is unlikely that the provider wouldn't allow for this. It is easily detectable, as we can simply test the data for changes, or know beforehand if none are allowed to be made.

In the case the risk does occur, we were ready to mitigate it by adapting the model we train on it as necessary.

Risk 3

Insufficient research may be due to a lack of interest or relevance in that field, or a lack of the necessary resources.

The severity of this is fairly low for us, as we expect little research in the field anyway. Our problem is niche, and we have set our schedule with the understanding that we will need to explore much of the domain ourselves, rather than base our work mostly on other projects. The occurrence of this is somewhat likely but fairly low, as music generation and music information retrieval, two similar and related fields, are popular. The detectability is low, as we will know if there is insufficient research by the lack of search results when we look for said research.

We planned to accept the risk if it did occur. It was expected that the risk would occur to some degree, and we made the necessary preparations in regard to our scope, timeline and schedule.

Risk 4

The ability of a team member to work effectively may become impacted by a variety of circumstances, some of which may be unavoidable, whether it be health issues or sudden responsibility, such as the need to take care of a family member.

The severity of this is high, as without enough team members the work will not be completed. However, the occurrence is low, as such debilitating events are rare. It would be easy to detect, by the lack of productivity of such a team member.

If the risk occurs, we accept it. There is little that can be done other than to offer the team member in question support, and adapt our schedule accordingly. If the situation became increasingly severe, we planned to contact senior staff as necessary.

Risk 5

The loss of work or data can be caused by carelessness by a team member, or by a failure in some service used to host it.

The severity of this is very high, as it can cause significant delays and may threaten the feasibility of the project. However, the occurrence is very low, as this threatens the service as well; poor service will mean people will stop using it, so it is in the provider's best interest to make the service as secure as possible. Its detectability is low, as missing work or data will be easily spotted in proofreading.

We planned to avoid this by opting for secure services, and making sure that no large piece of work is on any singular machine, but on one of the services mentioned in section 7.3.

Risk 6

The abandonment of libraries or resources can occur if they are no longer being used frequently.

The severity of this can be high, if we have already developed a system around such a library for which support is then deprecated. However, even if it does occur, we can typically shift to another library or resource as necessary, though this may take some time. The occurrence of this is very low because deprecation usually does not occur over such a short span of time, and it is very unlikely such an event would occur during our completion of this project. The detectability is also low, as we would notice if it were no longer available, we would observe some sort of error.

Accepting this means being ready to move to a different library or resource as necessary.

Risk 7

There may be a lack of resources if demand is too high, or the services are unavailable due to problems the provider may be facing.

The severity of this is high, as it significantly impacts our ability to train any machine learning models. The occurrence is 5, since demand for computational resources in general is high, and detectability is low, as we would know by being unable to train our models.

This risk can be mitigated by finding alternative options, such as other platforms that provide the necessary resources, beforehand in such an eventuality, and making the models as efficient as possible to reduce the load they put on a system, and thus the resources required.

7.5 Feasibility study

To ensure our project was viable, we looked at the feasibility of the project from many different perspectives, as described in the following sections.

7.5.1 Schedule feasibility

To understand the time required to complete the project, we identified the necessary tasks to be solved, and estimated the time required for each using Gantt charts, which can be seen in section 7.2. This was made with an understanding of the resources that would be available. Whilst we could not be certain, based on this estimation, we believed that it was feasible to complete the project on time. To ensure that this was the case, we held regular meetings to check on progress, and consulted our project supervisor for advice frequently. We also scheduled slack into each task, allowing us to adapt to any sudden needs.

In addition, we clearly identified any risks that may have delayed the project, along with contingency plans in case they come about to absolve them as quickly as possible. We discuss these risks in depth in section 7.4.

7.5.2 Operational feasibility

The specific problem our solution solves is to provide a music humanisation system. In this regard, we specify the exact components of such a system, in section 1.2. To ensure operational feasibility, we have made these aims clear and measurable and planned tests to ensure these aims had been achieved.

7.5.3 Technical feasibility

To ensure technical feasibility, we researched a number of similar projects, in section 2. Whilst our exact task has never been done, there are many successful projects in music generation, of which our project is an extension.

There are indeed some risks, which have been identified in section 7.4. Specifically, those that threaten technical feasibility are risks 1, 2, and 7. However, these risks have been evaluated and, as none of them have such a high-risk priority number to make the project infeasible, we have reason to believe that the project can be completed nonetheless.

7.5.4 Economic feasibility

The primary concern in economic feasibility was the cost of computation, as our work requires the use of high-end GPUs, to train the various machine learning models in section 5. For this purpose, we identified that we could use DCS batch compute[1]. In addition, we also found Google cloud compute[2], which provides $300 worth of free credits per trial.

7.5.5 Legal feasibility

Legally, there are no significant considerations. To ensure this, any time the work of any external individuals or groups is used, credit is given and we abide by the licenses for their work, where applicable.

7.6 Legal, social, ethical and professional issues

Due to the nature of the project, we use existing music work to train our models. We have dealt with any corresponding legal issues accordingly. The dataset that we use for this, ASAP, is available under the Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license. We abide by its terms; our task does not involve any commercial purpose, credit is given, and, since we do not redistribute the data, we have not distributed it under a different license following remixing or adaptation of the data.

Socially and ethically, the development of generative AI is an important topic of discussion. Some consider its development harmful to the industry, as it threatens the livelihood of creators. Some consider it beneficial in providing new ideas and means to grow the space. We don't believe our project involves the main concern of hurting new creators in any substantial way. None of the music created by the model directly endangers anyone. We understand that this is a difficult question to answer however, and as such have done our best to focus simply on the task at hand, and limit how much we impact other areas.

Throughout our work, the contribution of others has been referenced as necessary, and we have not collected any data that contains details that may identify individuals. As such, we have not encroached on anyone's anonymity or confidentiality. Thus, we believe professionally, our project does not pose any issue.

7.7 Evaluation

Whilst our methods are justifiable, it is important to reflect on what worked and what did not, and understand why this might be the case, so as to structure and manage future projects better.

[1] https://warwick.ac.uk/fac/sci/dcs/intranet/user_guide/batch_compute/
[2] https://cloud.google.com/compute

7.7.1 Risks

Out of our identified risks, only risk 1 was realised. From our initial research, we identified there were few datasets containing instructions. This resulted in the majority of our efforts in the first half of this project going towards creating preprocessing tools to alleviate this issue. However, we failed to notice that there was a significant data shortage for humanisation.

Whilst we did find a dataset that met our needs, it was not big enough for the models to train effectively. The primary issue was the lack of datasets that contained scores. Whilst datasets of performances were common, those that also contained the corresponding score for each performance were rare. Collecting the scores separately and combining them was largely unfeasible without the correct name of the piece attached to the performance. Even then, it was difficult, and would require writing a scraper from some web repository, and a huge amount of time to manually match all examples.

Nonetheless, this did not significantly reduce the scope of our project, as creating humanisation models was a secondary goal of our project.

7.7.2 Development methodology

Our agile approach worked well for us, since, as we expected, changes in requirements were frequent, and being flexible was important throughout the project. Particularly, even with the primary problem that we faced, the dataset, we were still able to adapt well and complete the project nonetheless.

There were however a number of aspects that did not work as well for us that can be evaluated to improve future work. First, our meetings were, in some weeks, awkwardly timed and not as productive as they could have been. Waiting a whole week to discuss problems discovered early into a week led to delays, and communication was lacking some weeks due to this. Rather than communicating what we were doing throughout the week as we should have, we used the meetings to catch up, which wasn't as effective. Having each team member talk about their work one by one in the weekly meetings was not engaging, and it was difficult to keep track of what everyone has mentioned.

It may have been beneficial to hold an additional meeting in the middle of the week, or be more willing to hold unplanned meetings throughout the week. The latter was difficult however, as everyone had different responsibilities, and it was not usually the case that everyone would be available at any particular time. Better communication throughout the week would have helped alleviate this problem as well somewhat.

Secondly, a lack of structure for our project was beneficial in some ways, but less so in others. To ensure that each team member had the space to explore their own ideas, we had to make sure that management was not too rigid; we wanted to avoid one person dictating what we would do, as none of us were experts in the domain. However, due to the lack of management, on occasion the exact task was not clear, and we ended up having multiple people working on the same problem.

7.7.3 Testing

Originally, we planned to perform comprehensive testing on the generations from our humanisation models. We aimed to follow our proposal for a 'Turing test' that we explained in section 1.1 to evaluate their performance beyond raw technical metrics. However, due to our dataset issues we did not find time to create the test and conduct the survey.

In future work, we would be more careful to enforce better planning to ensure some of this testing is performed. It is critical that the performance of humanisation models is judged by volunteers so we can provide a formal benchmark for others to compare against.

Conclusion

This section summarises our findings, evaluates our project as a whole, and highlights the key areas upon which future work should begin.

8.1 Evaluation of success

For this project, our first aim was to research existing work, which is addressed in section 2. Our primary objective was to create an automatic music humanisation system, composed of the following: a sectioning model, described in section 4.4; instruction extraction via optical music recognition and reverse engineering, described in sections 4.1, 4.2, 4.3 respectively; instruction interpretation model, described in section 4.5, using natural language processing; generation models for creating human sounding performances, described in section 5. We evaluate and summarise the project in section 6, addressing the final aim. Therefore in this project, we were able to achieve our set objectives, and deliver the specified systems.

8.2 Limitations

Among the limitations of each system, there is one common theme - the lack of data. Whilst we were able to develop preprocessing tools to help address these problems, these require datasets in themselves, which are easier to create than those directly required for a generation model. As such, we stress that the most important work for the future of this field is in developing the necessary datasets.

Other than affecting the performance and quality of the generated pieces from our machine learning models, it also makes testing systems rigorously difficult.

Specifically, more data consisting of scores, of classical piano pieces, and corresponding performances of those scores are needed. Whilst data of performances are abundant, they are either not labelled with the corresponding score, or do not correspond to a classical piano piece.

Creating such a dataset is difficult, as it requires a skilled musician, as well as a large amount of time. We do not have the resources to do so, but we welcome anyone interested in the field to tackle this, as it is the main priority in continuing this work.

8.3 Future work

In the future, other than building data, we highlight a number of improvements and ideas to expand upon for each system in section 6. For the music generation models, we focus on improving efficiency and performance. For the other systems, the improvements are more specific but similarly aim to improve the quality of the output of the system. Additionally, in the future we would perform the survey mentioned in section 1.2.1, to evaluate the music generation models we generated in a working user environment.

One system we would have liked to implement fully but could not fit into our schedule was the development of a web app to service as a user interface for ease of use of our solution. We could have also potentially used it for surveys to see the effectiveness of the project from an everyday user perspective. A mockup of this is given in Figure 8.1.

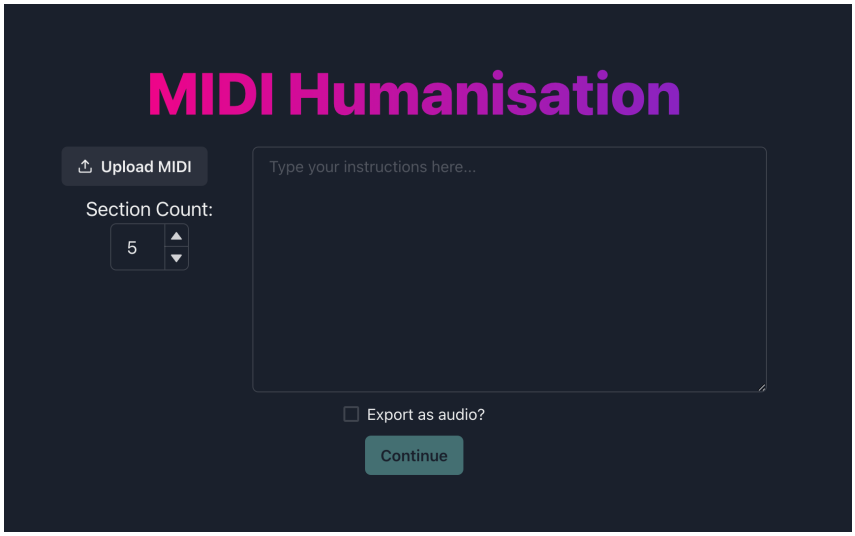


Figure 8.1: The design of the user interface.

Bibliography

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Banar et al., 2020] Banar, N., Daelemans, W., and Kestemont, M. (2020). Character-level transformer-based neural machine translation. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, pages 149–156.

[Barrington et al., 2010] Barrington, L., Chan, A., and Lanckriet, G. (2010). Modeling music as a dynamic texture. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18:602 – 612.

[Beltagy et al., 2020] Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

[Bittner et al., 2018] Bittner, R. M., McFee, B., and Bello, J. P. (2018). Multitask learning for fundamental frequency estimation in music.

[Black et al., 2022] Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al. (2022). Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Byrd and Simonsen, 2015] Byrd, D. and Simonsen, J. G. (2015). Towards a standard testbed for optical music recognition: Definitions, metrics, and page images. *Journal of New Music Research*, 44(3):169–195.

[Chou et al., 2021] Chou, Y.-H., Chen, I., Chang, C.-J., Ching, J., Yang, Y.-H., et al. (2021). Midibert-piano: Large-scale pre-training for symbolic music understanding. *arXiv preprint arXiv:2107.05223*.

[Correia-Silva et al., 2018] Correia-Silva, J. R., Berriel, R. F., Badue, C., de Souza, A. F., and Oliveira-Santos, T. (2018). Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.

[Dao et al., 2022] Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

[De Coster et al., 2021] De Coster, M., D'Oosterlinck, K., Pizurica, M., Rabaey, P., Verlinden, S., Van Herreweghe, M., and Dambre, J. (2021). Frozen pretrained transformers for neural sign language translation. In *18th Biennial Machine Translation Summit (MT Summit 2021)*, pages 88–97. Association for Machine Translation in the Americas.

[Dong et al., 2018] Dong, H.-W., Hsiao, W.-Y., Yang, L.-C., and Yang, Y.-H. (2018). Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[Edirisooriya et al., 2021] Edirisooriya, S., Dong, H.-W., McAuley, J., and Berg-Kirkpatrick, T. (2021). An empirical evaluation of end-to-end polyphonic optical music recognition. *arXiv preprint arXiv:2108.01769*.

[Eriksen and Rehman, 2022] Eriksen, T. and Rehman, N. u. (2022). Data-driven signal decomposition approaches: A comparative analysis.

[Eriksen and Rehman, 2023] Eriksen, T. and Rehman, N. u. (2023). Data-driven nonstationary signal decomposition approaches: a comparative analysis. *Scientific Reports*, 13(1):1798.

[Foscarin et al., 2020] Foscarin, F., McLeod, A., Rigaux, P., Jacquemard, F., and Sakai, M. (2020). ASAP: a dataset of aligned scores and performances for piano transcription. In *International Society for Music Information Retrieval Conference (ISMIR)*, pages 534–541.

[Friberg and Sundberg, 1993] Friberg, A. and Sundberg, J. (1993). Perception of just-noticeable time displacement of a tone presented in a metrical sequence at different tempos. *The Journal of The Acoustical Society of America*, 94(3):1859–1859.

[Gatys et al., 2015] Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[Gong et al., 2021] Gong, Y., Chung, Y.-A., and Glass, J. (2021). Ast: Audio spectrogram transformer.

[Goodfellow et al., 2020] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

[Hawthorne et al., 2019] Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., and Eck, D. (2019). Enabling factorized piano music modeling and generation with the maestro v3.0 dataset. In *International Conference on Learning Representations*.

[He et al., 2014] He, K., Zhang, X., Ren, S., and Sun, J. (2014). *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, pages 346–361. Springer International Publishing.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Huang et al., 2018] Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2018). Music transformer. *arXiv preprint arXiv:1809.04281*.

[Huang and Yang, 2020] Huang, Y.-S. and Yang, Y.-H. (2020). Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1180–1188.

[Katharopoulos et al., 2020] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Klapuri, 2003] Klapuri, A. (2003). Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *IEEE Transactions on Speech and Audio Processing*, 11(6):804–816.

[Kong et al., 2022] Kong, Q., Li, B., Chen, J., and Wang, Y. (2022). Giantmidi-piano: A large-scale midi dataset for classical piano music. *Transactions of the International Society for Music Information Retrieval*, 5(1).

[Lin et al., 2022] Lin, T., Wang, Y., Liu, X., and Qiu, X. (2022). A survey of transformers. *AI Open*.

[Lisena et al., 2022] Lisena, P., Meroño-Peñuela, A., and Troncy, R. (2022). Midi2vec: Learning midi embeddings for reliable prediction of symbolic music metadata. *Semantic Web*, 13(3):357–377.

[Liumei et al., 2021] Liumei, Z., Fanzhi, J., Jiao, L., Gang, M., and Tianshi, L. (2021). K-means clustering analysis of chinese traditional folk music based on midi music textualization. In *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pages 1062–1066.

[Longuet-Higgins, 1976] Longuet-Higgins, H. C. (1976). Perception of melodies. *Nature*, 263(5579):646–653.

[Louizos et al., 2017] Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. *Advances in neural information processing systems*, 30.

[Mao et al., 2018] Mao, H. H., Shin, T., and Cottrell, G. (2018). Deepj: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 377–382. IEEE.

[Na et al., 2015] Na, I., Kim, S., and Nquyen, T. (2015). A robust staff line height and staff line space estimation for the preprocessing of music score recognition. *Journal of Internet Computing and Services*, 16:29–37.

[Oore et al., 2020] Oore, S., Simon, I., Dieleman, S., Eck, D., and Simonyan, K. (2020). This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 32(4):955–967.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Radford et al., 2016] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Raffel, 2016] Raffel, C. (2016). *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, COLUMBIA UNIVERSITY.

[Raffel and Ellis, 2016] Raffel, C. and Ellis, D. P. W. (2016). Optimizing dtw-based audio-to-midi alignment and matching. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 81–85.

[Rebelo et al., 2012] Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A. R., Guedes, C., and Cardoso, J. S. (2012). Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1:173–190.

[Ríos-Vila et al., 2022] Ríos-Vila, A., Iñesta, J. M., and Calvo-Zaragoza, J. (2022). On the use of transformers for end-to-end optical music recognition. In *Pattern Recognition and Image Analysis: 10th Iberian Conference, IbPRIA 2022, Aveiro, Portugal, May 4–6, 2022, Proceedings*, pages 470–481. Springer.

[Schmidt-Jones, 2013] Schmidt-Jones, C. (2013). *Understanding basic music theory*. Rice University.

[Shatri and Fazekas, 2020] Shatri, E. and Fazekas, G. (2020). Optical music recognition: State of the art and major challenges. *arXiv preprint arXiv:2006.07885*.

[Singh et al., 2021] Singh, S., Wang, R., and Qiu, Y. (2021). Deepf0: End-to-end fundamental frequency estimation for music and speech signals. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 61–65.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Turing, 1950] Turing, A. M. (1950). Mind. *Mind*, 59(236):433–460.

[van der Wel and Ullrich, 2017] van der Wel, E. and Ullrich, K. (2017). Optical music recognition with convolutional sequence-to-sequence models. In *International Society for Music Information Retrieval Conference*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Wang et al., 2018] Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31.

[Wang and Haque, 2017] Wang, X. and Haque, S. A. (2017). Classical music clustering based on acoustic features.

[Yang et al., 2017] Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. (2017). Midinet: A convolutional generative adversarial network for symbolic-domain music generation.

[Yeh et al., 2010] Yeh, C., Roebel, A., and Rodet, X. (2010). Multiple fundamental frequency estimation and polyphony inference of polyphonic music signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1116–1126.

[Yu et al., 2022] Yu, B., Lu, P., Wang, R., Hu, W., Tan, X., Ye, W., Zhang, S., Qin, T., and Liu, T.-Y. (2022). Museformer: Transformer with fine- and coarse-grained attention for music generation. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Zeng et al., 2021] Zeng, M., Tan, X., Wang, R., Ju, Z., Qin, T., and Liu, T.-Y. (2021). Musicbert: Symbolic music understanding with large-scale pre-training. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 791–800.

[Zhang et al., 2022] Zhang, Y., Li, B., Fang, H., and Meng, Q. (2022). Spectrogram transformers for audio classification. In *2022 IEEE International Conference on Imaging Systems and Techniques (IST)*, pages 1–6.

[Zhuang et al., 2020] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning.