Informatics

Semantische Musik-Rekonstruktion mit Deep Learning

Verwendung von Graph-Neuronalen Netzen für die Erstellung von Musiknotengraphen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Grégoire de Lambertye
Matrikelnummer 12202211

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger
Zweitbetreuung: Univ.Lektor Dr. Alexander Pacha

Wien, 2. September 2024

Grégoire de Lambertye   Horst Eidenberger

Informatics

Music Semantic Reconstruction With Deep Learning

Learning How to Construct Music Notation Graphs With Graphs Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Grégoire de Lambertye
Registration Number 12202211

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger
Second advisor: Univ.Lektor Dr. Alexander Pacha

Vienna, September 2, 2024

Grégoire de Lambertye   Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Grégoire de Lambertye

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 2. September 2024

Grégoire de Lambertye

Acknowledgements

I want to express my sincere gratitude to everyone who has supported and guided me throughout the process of completing this thesis.

First and foremost, I am profoundly grateful to Doctor Alexander Pacha for his guidance, patience and invaluable advice throughout this thesis. Your insight and our conversations about music notation made me rediscover and appreciate its complexity. I will enjoy playing music all the more.

I am sincerely thankful to Professor Horst Eidenberger for supervising this thesis. I also thank Professor Jean-Marc Petit for his remarkable kindness and support throughout my research.

I owe my parents and family a special thanks for your unwavering support over the past years and during these last months. Your influence, especially our family musical evenings, led me to this musical topic for my master's thesis.

My heartfelt appreciation goes to my friends from Vienna, Lyon, and beyond. Your priceless support, discussions, and shared moments have been a source of strength and inspiration during the thesis and significantly contributed to my personal growth over the last few years.

Kurzfassung

Musik ist vergänglich; sie existiert nur kurz, bevor sie verhallt. Um Musik über die Zeit zu bewahren, wurden viele Notationssysteme entwickelt, die jedoch meist für die menschliche Interpretation gedacht sind. Die optische Musikerkennung (Optical Music Recognition, OMR)) beschäftigt sich mit der automatischen Erkennung dieser Notationen und deren Umwandlung in maschinenlesbare Formate. Die OMR-Prozesskette umfasst vier Hauptphasen: Bildvorverarbeitung, Erkennung von Musiksymbolen, semantische Rekonstruktion und Kodierung. Die semantische Rekonstruktion konzentriert sich darauf, die Bedeutungen der Musiknoten zu entschlüsseln, indem die Beziehungen zwischen den erkannten Musiksymbolen wiederhergestellt werden. In diesem Kontext wird die Semantik durch die Konfiguration und Anordnung musikalischer Grundelemente wie Vorzeichen, Notenköpfe und Fähnchen definiert, deren Gruppierung ihre Interaktionen und den intendierten Klang bestimmt. Studien betonen die grafische Natur von Musiknoten und führen das Konzept des Music Notation Graph (MuNG) ein, in dem musikalische Primitive als Knoten und deren Beziehungen als Kanten dargestellt werden. Diese graphische Struktur bietet vielversprechende Möglichkeiten für den Einsatz von Graph Neural Networks (GNN). In der vorliegenden Masterarbeit wird die Anwendung von GNNs zur semantischen Rekonstruktion in der OMR untersucht. Es wird eine innovative Pipeline vorgeschlagen, die GNNs in OMR integriert, und es werden Herausforderungen wie die Bewertung und der Vergleich der OMR-Ergebnisse sowie die Definition von MuNGs diskutiert.

Abstract

Music is fleeting by nature, existing briefly before fading away. Yet more than one wants to preserve it over time and many music notation systems have been developed with this intention. However, most of these formats have been designed for humans to read them. Optical Music Recognition (OMR) is a research field dedicated to investigating how to computationally read music notation and create computer-readable scores from traditional scores. The typical OMR pipeline is divided into four stages: Image pre-processing, Music object detection, Semantic reconstruction, and Encoding. The third stage of the pipeline aims at reconstructing the semantics of the music notation, re-establishing connections between the objects detected previously. This work focuses on scores written in the most common notation system called Common Western Music Notation (CWMN). In this context, the semantics are defined by the configuration of the musical primitives (accidental, noteheads, or flags), how they are grouped and arranged defines their interactions and how the music should sound. Some research exhibits the graph-like property of the music and introduces the concept of Music Notation Graph (MuNG): graphs constructed with the music primitives as nodes and their relations as edges. This graph structure makes it a candidate for leveraging the power of Graph Neural Networks (GNN). This master thesis investigates how GNNs can be used to perform the music semantic reconstruction. We propose a novel pipeline for using GNNs in OMR and discuss a few unsolved problems of the field like how to measure and compare the output of an OMR system or how to define MuNGs.

Contents

CHAPTER 1

Introduction

"Unless sounds are held by the memory of man, they perish, because they cannot be written down." St. Isidore of Seville[1]

1.1 Saving Music From Oblivion

Music is fleeting by nature, existing briefly before fading away. Yet composers strongly desire to immortalise and share their musical creations beyond that moment. For a long time, oral transmission was the only means to preserve music. However, music notations progressively appeared to offer a more sustainable preservation. Music notation, as we define it, is a system of symbols that aims to encode enough information to enable the music to be reproduced. Nowadays, the most commonly used music notation is called Common Western Music Notation (CWMN). It is the result of centuries of evolution: the earliest form of music notation comes from about 1400 BCE [2]. It is a collection of cuneiform inscriptions on clays that describe a religious hymn and how to play it on a Sumerian lyre. Other notations were developed in Ancient Greece [3], in the Byzantine Empire [4], and in the Middle East during the 13th century [5]. In Europe, during the 9th century, Aurelian of Réôme wrote Musica Disciplina, the earliest writing about music theory in the Occident. With this book, he established the base of our modern notation using symbols known as neumes to define pitches [6]. Later, many additions were made to make the notation more versatile and to allow capturing a wider range of properties: rhythms were added in the 13th century, and dynamics in the 17th. Even today, music notation keeps evolving to account for new developments and to capture non-traditional music.

Music notation is a language that needs to be embodied in a document called a score. Music scores can dress diverse appearances. The first scores were handwritten, but typeset scores appeared long before the advent of computers; in the middle of the 15th century, wood blocks and metal plates were used to partially or entirely print the music notation [7]. Later, with the widespread adoption of computers, many software programs [8] [9] [10] [11] emerged to facilitate the creation of scores. Known as scorewriters or music notation programs, these applications offer a convenient way to produce typeset, digital scores. They also enable users to listen to their creations by synthesising them, creating an audio representation of the music score. To illustrate the diversity of forms music scores can take, we present a comprehensive taxonomy illustrated in Figure 1.1. Scores with both handwritten and typeset typography are relatively common; e.g. many composers write their music manually on paper with preprinted staff. We claim that music notations and their scores are intended to be read either by humans or machines. In theory, some score formats, like MusicXML, can be read by both, but they are intended to be read by machines. Some character combinations lead to uncommon scores, e.g. typeset physical scores intended to be read by machines can be piano rolls. Conversely, a handwritten digital score intended to be read by machines can be the input of a music notation software like StaffPad [12], where users draw the music notation manually on a graphic tablet, and the software *reads* it. Knowing that music notation can't preserve the music without loss, reading involves the exhaustive recovery of the musical information described in a score. The performer must fill the gaps between the notation and the actual music to play the music. Filling these gaps is called i*nterpreting*; multiple interpretations can correspond to the same music score.

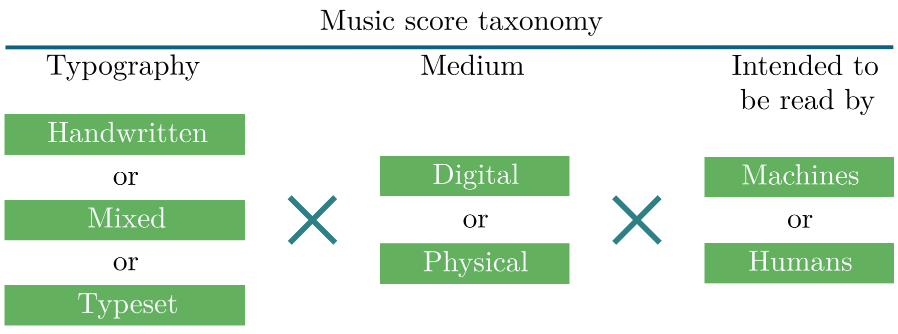


Figure 1.1: Music score taxonomy.

.

Music can not only be written to scores; it can also be recorded and preserved through audio encodings. Vinyl, CDs, or digital files like MP3 or WAV encode the physical sound wave over time. Audio encodings are mainly used for playback. Unlike music notation, the playback of an audio encoding does not require interpretation and always sounds the same (assuming the same room and equipment for playback). The most common formats used to preserve music are digital audio encodings and music scores intended for human reading, but other formats present particularly interesting properties too.

We define computer-readable scores as documents that contain a structured, symbolic encoding of music which can be read by a computer. Computer-readable scores can also be read and modified with music notation software. The most common formats are Musical Instrument Digital Interface (MIDI) [13], Music Encoding Initiative (MEI)

[14], and musicXML [15]. Computer-readable scores offer numerous advantages for composers, musicians, and researchers. They can easily be transformed (synthesised) into audio encodings or engraved into other score formats, such as tablatures for guitarists. They enable easy modifications, such as transposition, which involves shifting a piece of music to a different key. This is particularly beneficial for accommodating the vocal ranges of different singers or the tuning of various instruments. Additionally, these scores enhance music querying capabilities, allowing for advanced search and analysis of musical information. For instance, one could identify recurring motifs or themes across different works by a composer or within a particular genre, a precious feature for musicological research. The potential applications of these scores are vast, promising significant advancements in how we create, study, and interact with music.

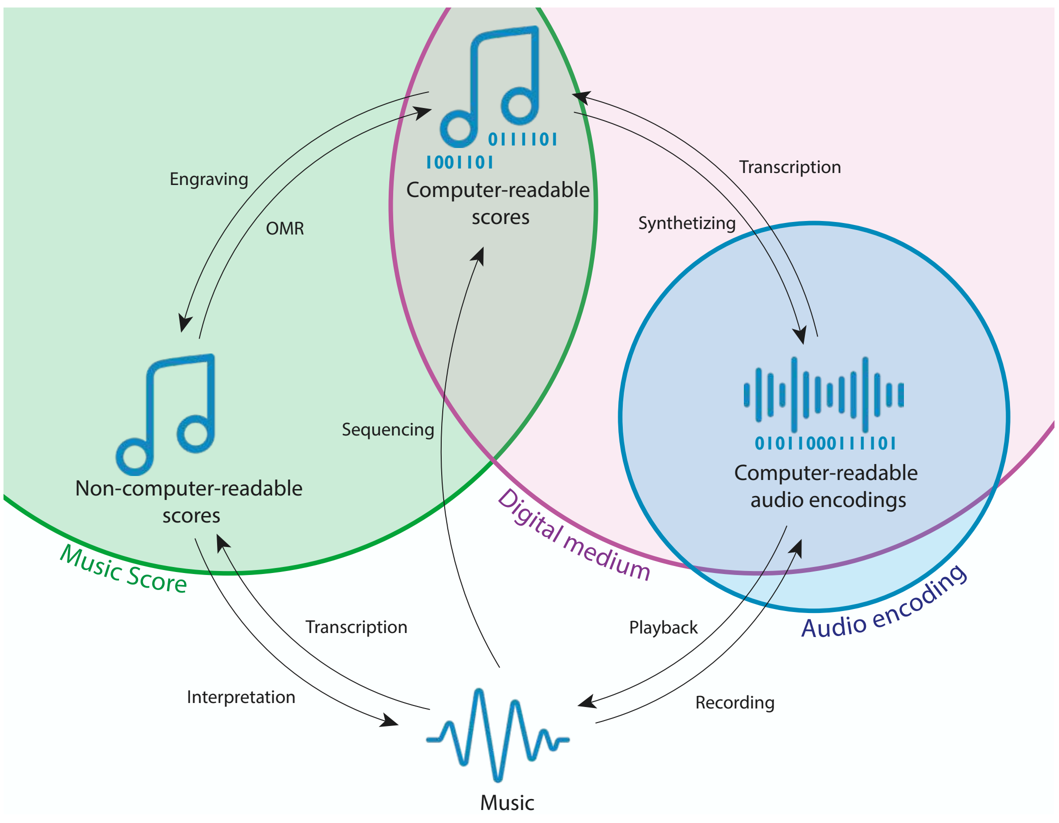


Figure 1.2: Transitions between music and some preservation means.

Figure 1.2 illustrates the transition between some music preservation formats and the music. Some transitions are very uncommon and, therefore, not represented. In this figure, computer-readable scores can be obtained by sequencing the music. This usually involves saving a sequence of instructions into the appropriate format, like a MIDI keyboard with the corresponding software would. Computer-readable scores can also be obtained by transcribing audio encodings. This involves more advanced techniques traditionally based on analysis of the frequencies present in the sound wave. The last transition that gives computer-readable from non-computer-readable scores is the goal of a field of research called Optical Music Recognition (OMR).

1.2 Optical Music Reconstruction

In *Understanding Optical Music Recognition*, Calvo et al. [16] address a notable issue within the field of OMR: the lack of a universally accepted definition for the term. This ambiguity in definition has led to inconsistencies in the scope and goals of research within the field. To provide clarity and a cohesive direction for future studies, Calvo et al. propose the following definition: “*Optical Music Recognition is a field of research that investigates how to computationally read music notation in documents*”. With the previous section, we can refine this definition: Optical Music Recognition is a field of research that investigates how to transform non-computer readable scores into computer-readable scores.

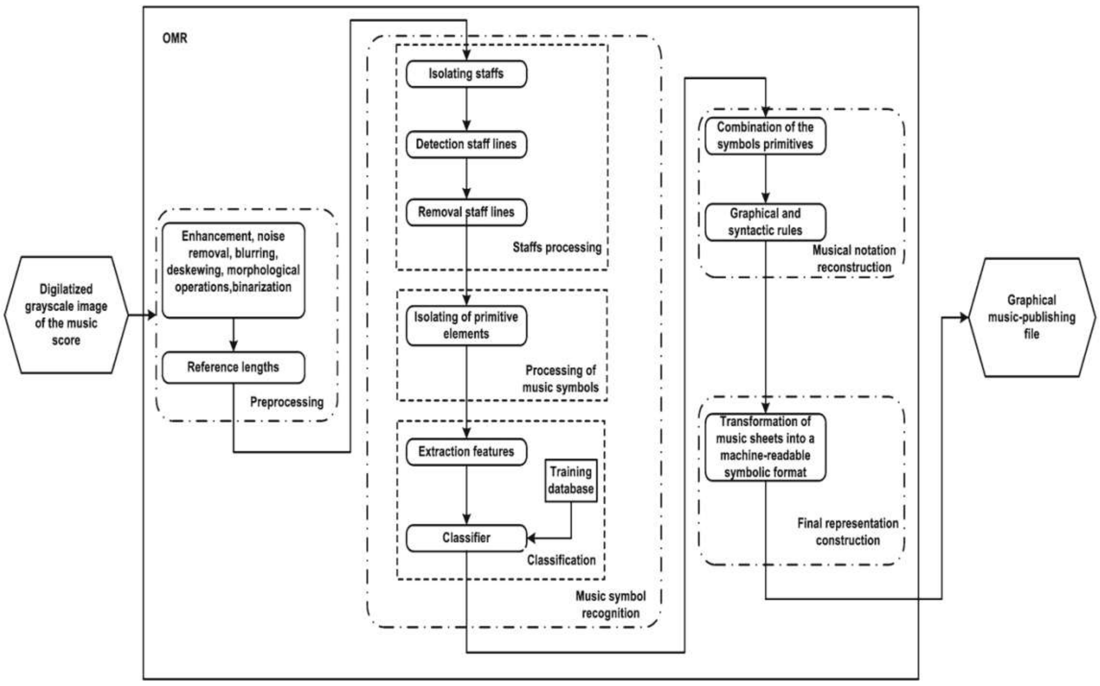

OMR
Digitalized grayscale image of the music score
Enhancement, noise removal, blurring, deskewing, morphological operations,binarization
Reference lengths
Preprocessing
Isolating staffs
Detection staff lines
Removal staff lines
Staffs processing
Isolating of primitive elements
Processing of music symbols
Extraction features
Training database
Classifier
Classification
Music symbol recognition
Combination of the symbols primitives
Graphical and syntactic rules
Musical notation reconstruction
Transformation of music sheets into a machine-readable symbolic format
Final representation construction
Graphical music-publishing file

Figure 1.3: Typical architecture of an OMR system as proposed by Rebelo et al. [17]

The inherent complexity of music notation, which encodes pitch, duration, tempo, and rhythm into a two-dimensional representation, presents a distinct set of challenges for accurate transcription. Rebelo et al. [17] propose to compartmentalise the OMR pipeline into four stages, illustrated in Figure 1.3:

1. Image pre-processing,
2. Recognition of musical symbols,
3. Reconstruction of the musical information in order to build a logical description of musical notation or semantic reconstruction,
4. Construction of a musical notation model to be represented as a symbolic description of the musical sheet.

Image pre-processing uses standard techniques to clean the input and make the content stand out from the background. The second stage aims to find and classify the relevant symbols of the input image. These symbols often consist of so-called musical primitives (e.g., accidental, notehead, or flag) that aim to represent the atomic symbols of the music notation. Figure 1.4 partially illustrates this alphabet. The output of the Music object detection stage is the class and position (often as bounding boxes) of the primitives. In OMR, the semantics are defined by the configuration of musical primitives (how they are grouped and arranged) and their interactions with others. On their own, musical primitives don't have any musical meaning. The third stage aims at reconstructing the semantics of the music notation, re-establishing connections between the detected objects, emphasising their relationships, and facilitating the transformation into a computer-readable format. This stage lacks sufficient research, but recent deep learning approaches have emerged as promising avenues for enhancing this particular pipeline stage. The last stage involves encoding the music into computer-readable scores.

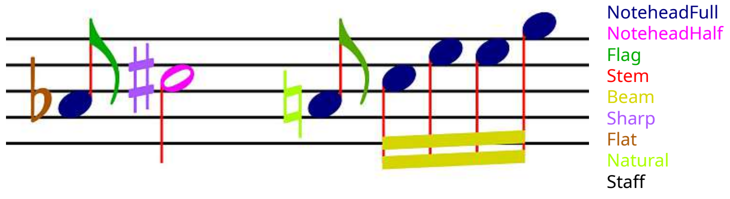


Figure 1.4: Small music score where music primitives are shown in colours.

1.3 Research Questions

This work aims to investigate the use of Graph Neural Network (GNN) in the third stage of the OMR pipeline. It focuses on music scores in Common Western Music Notation and explores how Graph Neural Networks can be used to reconstruct the relationships between musical primitives. The spatial relationships between these primitives, which are crucial for reading music, can be represented as a graph and used as input for a GNN. In the literature, this graph of musical primitives and their relationships is referred to as Music Notation Graph (MuNG) [18]. The semantic reconstruction task corresponds to a link prediction or link classification task for which GNNs have demonstrated strong capabilities.

This work investigates the following four research questions:

1. How suitable (5% of Music Error Rate (MER) or lower, see Section 4.2 for details) are GNNs to solve the music semantic reconstruction stage?

   5% of MER corresponds to the best value reached by Baro et al. [19], who introduced the metric. No prior research enables to set a threshold to define when a system is suitable for OMR. However, some applications, like snippet querying (finding a score based on a snippet), don't require a perfect comprehension of the score [20]. We consider 5% of MER a baseline, but an ideal system would achieve a MER of 0%.

2. How does such a model handle different input configurations, particularly in terms of musical complexity (amount of musical information on a score like multiple instruments, accompanied melody...)?

3. How well do GNN-based approaches for semantic reconstruction scale with an increasing number of input primitives?

4. How does the GNN initialisation (fully connected, based on explicit rules) impact the link prediction?

1.4 Organisation

The structure of this work is as follows: Chapter 2 provides background information on how to read music and an introduction to graph neural networks. Chapter 3 presents the datasets used during the implementation phase and discusses the challenges encountered in creating a dataset for Optical Music Recognition. Chapter 4 covers the main implementation, detailing the key challenges faced throughout the process. This chapter also discusses the metrics used and the results obtained. Finally, Chapter 5 concludes the thesis, summarising the findings and suggesting directions for future work.

CHAPTER 2

Background

2.1 How to Read Music?

Common Western Music Notation (CWMN) represents music using standardised symbols interconnected by relations. This section outlines how CWMN functions and why reading it can be challenging. It introduces essential concepts relevant to this work but simplifies and omits many aspects of the notation. For a more detailed and precise introduction to music notation, refer to Schobrun's step-by-step guide [21].

Music scores are read from top to bottom and from left to right. They rely on staves: five parallel lines resembling a line of text. The other symbols are displayed on the staves (see Figure 1.4) and may form the *notes*. A note is a sound with a defined pitch, onset (when to play it), duration and timbre. We can simplify the score reading by assuming each notehead corresponds to a note. The vertical position of noteheads indicates the pitch. The pitch of a note can also be modified by other primitives, e.g., accidentals such as sharp or flat. The duration of the note is defined in beats or fractions of, with the ”beat” defining the tempo of the composition. The duration of a note is defined by a combination of notehead type, presence of a stem, and optional flags or beams, among other things. The use of more advanced indications can complexify the scores. Some primitives specify how to articulate a note (e.g., pinched, with hard or soft attacks). Dynamics indicate broader behaviours for multiple notes, such as loudness or tempo acceleration. The number of primitives in CWMN is not well defined. Some music fonts list hundreds of musical glyphs (See 3.2.1). CWMN has no theoretical limitation regarding the density and the number of primitives that can be engraved on a score (however there is a practical limitation, governed by the available space on the used piece of paper). This notation also allows for symbols to overlap, which can make reading a score very complex. Figure 2.1 illustrates what a very dense score can look like with an extract of Faerie's Aire and Death Waltz. This composition is close to a parody of composition and is not meant to be played. Nevertheless, the notation is correct and audible versions of this score exist.

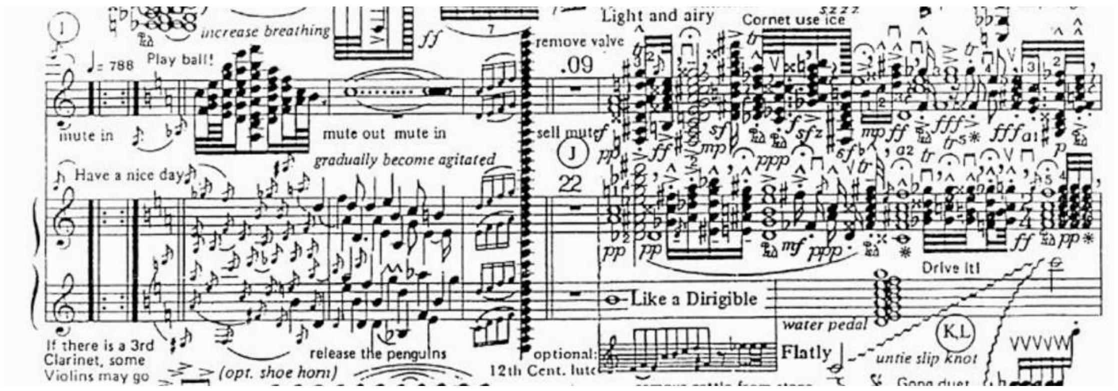


Figure 2.1: Extract of Faerie's Aire and Death Waltz by John Stump

The complexity of music scores is often discussed in terms of *structural complexity*. Calvo et al. define four levels of structural complexity illustrated in Figure 2.2 [16]. The structural complexity of a score mostly depends on the number of *voices* notated per stave. A voice refers to a single melodic line in music; the melody is usually handled by the most significant voice. The simplest structural complexity is called monophonic; one single note (per staff) is played at a time. Homophonic scores allow for multiple notes to be played at a time but still encode a single voice. In polyphonic scores, different voices can be engraved in a single staff. Pianoform scores encode multiple voices and require several staves, which demonstrate significant structural interactions and are often way more complex than polyphonic scores. In contrast to polyphonic scores, they often can't be broken down into a series of monophonic scores.

To structure a score rhythmically, the measures offer manageable segments of a consistent number of beats. Each measure contains a specific number of beats, which are determined by the time signature of the composition that is usually fixed for a whole music piece (see [21]). Measures are visually represented by vertical bar lines that divide the staff into beat-equal sections. This segmentation facilitates reading and aids in the composition process by providing a clear structure for arranging rhythmic patterns, melodies, and harmonies.

Regardless of the complexity of the score, reading the music always involves recognising the symbols and the relations that connect them to compute each note's pitch, duration and onset. The set of symbols and their relations can be viewed as a graph. In OMR, the process of reading a score can be cut by task and handled by different systems. Recognising the symbols is handled by music object detectors and recovering the edges corresponds to the semantic reconstruction stage. Considering the resemblance to a graph, this stage can be called a link prediction or link classification task. Graph Neural Networks have proven to be valuable tools for this task.

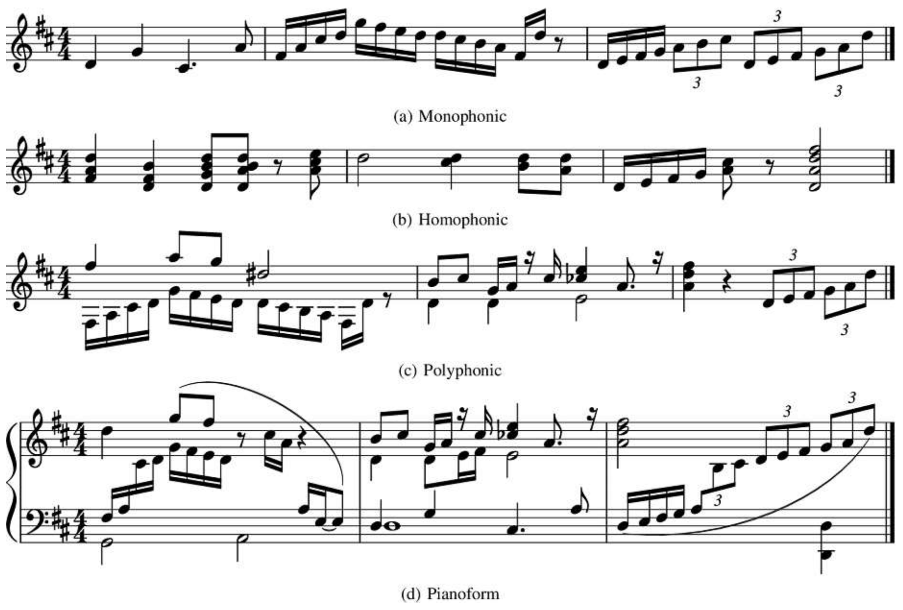


Figure 2.2: Examples of the four categories of structural complexity [16].

2.2 Graph Neural Networks

2.2.1 Reasoning on Graph-Structured Data

Graph Neural Networks are a class of machine learning models introduced by Gori and Scarselli in 2005 [22]. Unlike traditional neural networks that operate on regular grid-like structures such as images (2D grids) or sequences (1D grids), GNNs can directly process graph-structured data. A graph is defined as "*a finite, nonempty set V of objects called vertices [or nodes] (the singular is vertex [or node]) and a set E of 2-element subsets of V called edges*" [23]. Graphs are usually noted as $G = (V, E)$, and the vertices associated with edges being called their endpoints. *Links* and *edges* are synonyms. While "edges" is the preferred term in the mathematical literature, "link prediction" is more commonly used when referring to the task of predicting these connections. In this thesis, "links" and "edges" are used interchangeably. GNNs take graph-structured data as input and iteratively update the representations of their nodes with aggregated features from their neighbours.

This thesis does not aim to provide an exhaustive explanation of how Graph Neural Networks work; however, the key aspects are briefly summarised below. This section heavily relies on the paper Everything is Connected by Veličkovič [24]. The following assumes an unweighted directed graph with no edge features for simplicity.

The features of a node $u \in V$ is a vector $x_u \in \mathbb{R}^k$, where k, denotes the number of features. The graph representation usually relies on a feature matrix X and an adjacency matrix A.

$$X \in \mathbb{R}^{|V| \times k}, \quad X = \left[x_1, x_2, \ldots, x_{|V|}\right]^T \tag{2.1}$$

$$A \in \mathbb{R}^{|V| \times |V|}, \quad a_{uv} = \begin{cases} 1 & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases} \tag{2.2}$$

The matrix form of the graph imposes an order to the nodes, that contradicts the concept of nodes and edges being unordered in a graph. To ensure this arbitrary order does not impact the operation of a GNN, GNNs must comply with two principles: Invariance and Equivariance. The invariance property means the operation output stays still regardless of the order of the nodes, and the equivariance ensures some properties, such as the conservation of symmetry. The permutation of the result of an equivariant operation is equivalent to applying this operation on permuted inputs. Given a permutation matrix P, these two properties can be expressed as follows:

$$f(PX, PAP^T) = f(X, A) \quad \text{(Invariance)} \tag{2.3}$$

$$F(PX, PAP^T) = PF(X, A) \quad \text{(Equivariance)} \tag{2.4}$$

Assuming f and F do not change the adjacency matrix and return graph or node-level outputs.

The neighbourhood of a node u is the set of nodes N_u such as $(v, u) \in E$ for a directed graph. The neighbourhood of every node can be defined as X_{N_u}.

$$X_{N_u} = \{x_v \mid v_u \in N_u\} \tag{2.5}$$

The representation of the node u in a GNN output is obtained with a function:

$$h_u = \Phi(x_u, X_{N_u}) \tag{2.6}$$

Or for the whole node set:

$$F(X) = \left[h_1, h_2, \ldots, h_{|V|}\right]^T \tag{2.7}$$

Equation 2.7 [24] implicitly assumes some information about the neighbourhoods is known by F. The node neighbourhoods are required as input of Φ to compute the node representation. F could be rewritten $F(X, A)$ with A the adjacency matrix containing the neighbourhoods reconstitution.

If Φ is permutation invariant inX_{N_u}, meaning the order of the node's neighbourhood doesn't change the output of Φ, then F will be permutation equivariant. Φ is the heart of the GNNs, it defines how to aggregate and update a node representation and defines a GNN layer.

Bronstein et al. [25] claims most of the GNNs can be classified into three *flavours*: *Convolutional*, *Attentional* and *Message Passing (MP)*.

$$h_u = \Phi(x_u \oplus_{v \in N_u} \Psi(x_v)) \quad \text{(Convolutional)} \tag{2.8}$$

$$h_u = \Phi(x_u \oplus_{v \in N_u} a(x_u, x_v)\Psi(x_v)) \quad \text{(Attentional)} \tag{2.9}$$

$$h_u = \Phi(x_u \oplus_{v \in N_u} \Psi(x_u, x_v)) \quad \text{(Message Passing)} \tag{2.10}$$

Where Ψ and Φ are traditional neural network -e.g. $\Psi(X) = ReLU(WX + b)$ and $\oplus$ is a permutation invariant operator, such as the mean, sum or max function. In the Attentional equation, a is an attention mechanism, and $a(x_u, x_v)$ is a scalar. This set of equations defines what a single layer does. For multi-layer models, the matrix $X^{(k=0)}$ used as input for the first layer is set to $X^{(k+1)} = [h1, h2, \ldots, h|V|]$ T to be fed into the next layer. The matrix A is never updated.

The choice of these flavours might seem a bit arbitrary. First, because some layers don't fit into one of these categories [26]. Then, the limit between Attentional and Message-Passing GNN is controversial. In the MP equation, ψ can be defined as $\psi(x_v,\ x_u) = a(x_u, x_v)\theta(x_v)$, with θ a neural network. This form corresponds to an Attentional GNN as defined by Veličkovič; every MP GNN can be rewritten as an Attentional GNN. The other transformation is also always possible as the attention mechanism may be a neural network. $a(x_u, x_v)\psi(x_v)$ can be rewritten as $a(x_u, x_v)\psi(x_v) = \theta(x_u, x_v)\psi(x_v) = \gamma(x_u, x_v)$ with θ and γ being neural networks. The last equivalence relies on the fact that multiple neural networks can be combined into one. Every Attentional GNN can be rewritten as an MP GNN.

Veličkovič [24] states that the expressiveness of the GNNs increases through the three categories of layers at the cost of learning stability and interpretability. The decrease in interpretability is not that evident. Interpretability is defined by Zhang et al. [27] as the "*ability to provide explanations in understandable terms to a human*". Even the simplest flavour of GNN (Convolutional GNN), is the generalisation of a neural network. These systems are by nature black boxes [27] and therefore not interpretable. We can state that no GNN is, by nature, interpretable, and the pertinence of comparing their interpretability is hence questionable. In Section 4.7, the comparison of three GNNs shows learning stability can be an issue when using GNNs, and even Convolutional Layers might be unstable.

GNNs have proven strong capabilities in learning specific graph-related tasks on various application domains. Three main tasks can be achieved with GNNs:

- **Node classification** corresponds to the tasks where a node feature is the target. A classifier is directly applied to the node representation h_u to predict the node's feature. A famous example is the prediction of the category of a scientific paper in the CORA dataset [28].
- **Graph classification** corresponds to the task where the target is a graph property. An invariant operator is applied to every node representation $g = \bigoplus_{u \in V} h_u$ and a classifier is then applied to g to predict the target value. A known example predicts molecules' properties like their toxicity [29].
- **Link prediction** corresponds to the task where an edge feature is the target. A common property to predict is the existence of a link. To perform such tasks, a classifier learns over the concatenation of the two candidate nodes' embeddings h_u, h_v. The prediction of drug-drug interaction [30] or this master thesis correspond to this task.

2.2.2 A Closer Look at Some Layers

There exist many ways to define a suitable function that could be a GNN layer. Three layer implementations have been selected for the implementation phase of the thesis. They are described hereafter. GNN layers and GNNs are discussed indiscriminately in the literature. New layers are presented in GNNs to test their capacities, therefore, a single name is given to the GNN and its layer(s).

Geometric Graph Convolutional Neural Network (Geo-GCN)

Geo-GCN is the short name for Geometric Graph Convolutional Neural Network, a convolutional GNN proposed by Spurek et al. [31]. They state that classical Graph Convolutional Networks (GCN) are limiting because they cannot consider an ordering of nodes in a neighbourhood. What they call order has more to do with the importance one node can have for another. For some graphs, a natural order can be found. For example, nodes that are spatially close to the target node are usually more important than the further distant ones. The particularity of geo-GCN is to process spatial information differently from the other node features.

To each node u, is associated $p_u \in \mathbb{R}^t$, a vector of coordinates of dimension t. p_u corresponds to the spatial position of u. Unlike the node features, $P \in \mathbb{R}^{|V|xt}, P = p_1, p_2, \ldots, p_{|V|}{}^T$, is fixed across layers .

The layer equation is presented as follows in the paper:

$$\overline{h_u}(u, b) = \sum_{v \in N_u} \text{ReLU}(a^T(p_j - p_i) + b)h_v \tag{2.11}$$

Where $a \in \mathbb{R}^t$, and $b \in \mathbb{R}$ are trainable.

The scalar obtained after the sum function is used to weight the influence of the feature vectors h_j in the computation of h_i. To obtain the final representation of a node, a Multi-Layer perceptron (MLP) is applied on $\bar{h_u}$.

Using the previous equation format, the equation of the layer can be rewritten as:

$$h_u = \Phi\left(\oplus_{v \in N_u \cup u} \Psi(p_v - p_u) x_v\right) \tag{2.12}$$

With Ψ and Φ being multi-layer perceptrons with the ReLU activation function, and $\oplus$ the sum aggregator.

Graph Attention Network

Graph attention networks have first been descibed by Veličkovič et al in 2018 [32]. The key aspect of this GNN is to rely on a self-attention mechanism. Self-attention is a technic that allows neural systems to dynamically focus on and weight elements from an input sequence. In other words, the system learns what elements of the input are the most relevant and weights them accordingly. In their paper, the authors claim their GNN to be (1) efficient since the attention architecture is parallelizable across node-neighbours pairs. (2) Largely applicable since it can be used on graph nodes having different degrees and (3) directly applicable to inductive learning problems.

The authors claim that at least one learnable linear transformation is necessary to gain sufficient expressive power for converting the input features into higher-level features. Therefore, all node features are transformed with a learned matrix $W \in \mathbb{R}^{F' \times F}$. F' and F are respectively the dimensions of the output and the input of a layer. The importance of node v to node u, referred to as the *attention coefficients*, is e_{uv}, is determined by the attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$:

$$e_{uv} = a(W\overrightarrow{h_u}, W\overrightarrow{h_v}), \text{ for } v \in \{N_u, v\} \tag{2.13}$$

To ensure meaningful coefficient comparison across different nodes, a softmax function is applied to normalize the coefficients of a node across neighbourhoods.

$$a_{uv} = \text{softmax}_v(e_{uv}) = \frac{\exp(e_{uv})}{\sum_{k \in \{N_u, u\}} \exp(e_{uk})} \tag{2.14}$$

In the default configuration of the layer, the attention mechanism is a single-layer feedforward neural network, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, and a LeakyReLU activation function (with negative input slope $\alpha = 0.2$).

To stabilise the learning process, they eventually employ a multi-head attention mechanism (see Attention Is All You Need [33] for details). For K independent attention mechanisms, using our previous notation, the equation corresponding to a GAT layer is:

$$h_u = \Phi\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{v\in\{N_u,u\}} a^k_{uv}W^k x_v\right) \tag{2.15}$$

GraphSAGE

GraphSAGE is a very popular Message Passing GNN introduced in 2017 by Hamilton et al [34]. The novelty of this layer is that it is a general inductive framework, which means, unlike previous approaches, it efficiently generates node embeddings for previously unseen data. Instead of training individual embeddings for each node, it learns a function that generate embeddings by sampling and aggregating features from a node's neighbourhood.

In practice, a graphSAGE layer corresponds to the default message passing GNN layer which equation was presented in the previous section: Equation 2.10.

2.2.3 PyTorch Geometric

All experiments that were conducted as part of this thesis were implemented in Python, using the PyTorch framework with the PyTorch Geometric (PyG)[35] extension. The code is available on GitHub[1].

PyTorch is an open-source machine learning library widely used for deep learning applications. Initially released in 2016 by Meta AI (formerly Facebook), it provides a flexible and intuitive interface for defining and training neural networks. It also offers GPU acceleration and is one of the most popular libraries of this type. Released in 2019, PyG is specifically designed for deep learning on irregularly structured data such as graphs. It provides a comprehensive suite of tools for creating and training Graph Neural Networks and efficient implementations of various GNN layers.

[1]https://github.com/GregoireLamb/music_semantic_reconstruction.git

CHAPTER 3

Datasets for OMR

3.1 Datasets

The OMR field is relatively small, and constructing music notation graphs is just one of several possible approaches to reconstruct the semantics of the notation. As a result, only a limited number of datasets are available, each adhering to its own principles for encoding and defining a ground truth. This chapter presents the datasets used to answer the research questions and discusses the necessary choices to make for building such datasets.

3.1.1 Muscima++

MUSCIMA++ is a dataset created in 2017 and updated in 2019 [36]. It was built on top of the CVC-MUSCIMA Database [37] by adding to 140 handwritten scores their music notation graphs. Muscima++ aims to offer an adequate dataset and ground truth for benchmarking OMR systems. The whole dataset contains a total of 102’914 elements and 144’386 relations. It uses a very precise set of 115 labels to categorise the music primitives and counts 218 types of relations. A relation type is defined by the labels of the primitive involved in the relation, e.g. ‘notehead’-‘stem’. All the labels, their number of appearances, and the 80 most present relation types with their number of appearances are listed in Appendix A (Tables A.1 and A.2). The 140 scores of the dataset are all handwritten and their structural complexity varies from monophonic to polyphonic. Each score corresponds to a single page and is accompanied by its picture. Figure 3.1 shows an example of a polyphonic score from this dataset.

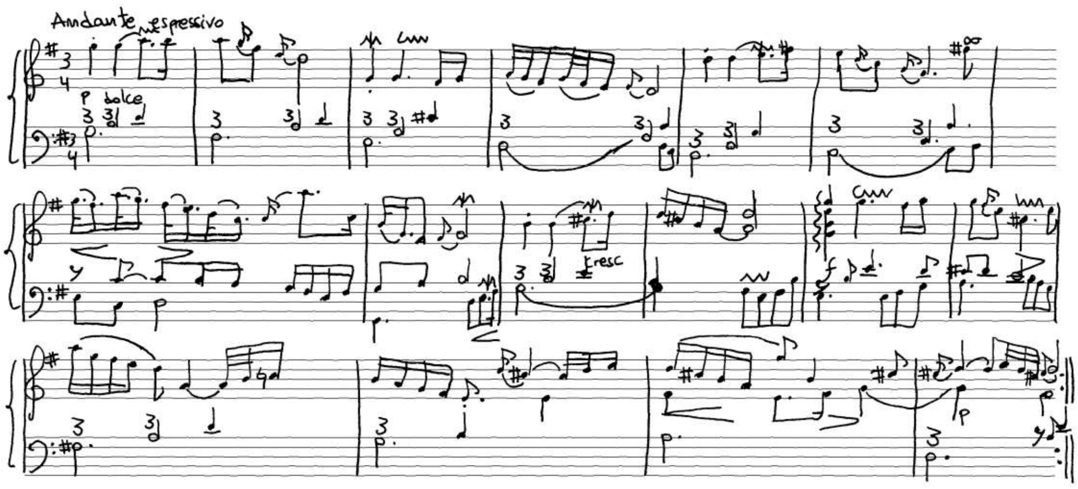


Figure 3.1: Polyphonic score extracted from Muscima++. By default the scores appear white with a black background, the colours have been swapped for visibility

This dataset comes with some extra information, such as how it has been annotated and two predefined test splits. The scores have been written by 50 musicians. One split (named dependant) asserts that the scores of a writer are distributed in training and test sets. The other test split (*independent*) is independent of the writers. We used version 2.1 of Muscima++ and its "independent" test.

3.1.2 Musigraph

Musigraph is a dataset developed by Baro et al. in 2022 [19]. The authors found Muscima++ limiting due to its small number of scores. To address this, they created a larger dataset with 15,218 scores. Each score represents a single measure of a Beethoven symphony simplified to be monophonic. Musigraph contains 370,404 elements and 253,623 relations, uses 18 music primitive categories and 11 link types. Details about these categories and link types are provided in annexes A (Tables A.3 and Table A.4). All the scores are monophonic and correspond to a single measure. Figure 3.2 shows an example of a score from Musigraph.

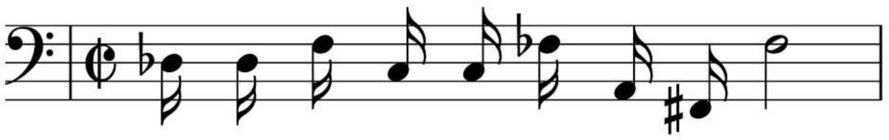


Figure 3.2: Example of score from Musigraph

This dataset was initially developed for a specific paper, and the meta-information only specifies the scorewriter used to generate it, along with a test–train–validation split. This dataset contains an unknown number of errors, but its simplicity makes it an interesting base for developing a model. Figure 3.3 presents a heatmap with marginal histograms illustrating the size and repartition of the 15'218 MuNGs corresponding to the scores. The graphs have small numbers of nodes roughly proportional to their number of links. Most scores have between 7 and 40 primitives, and the average node degree is 1.37, which corresponds to the average number of relations a primitive makes.

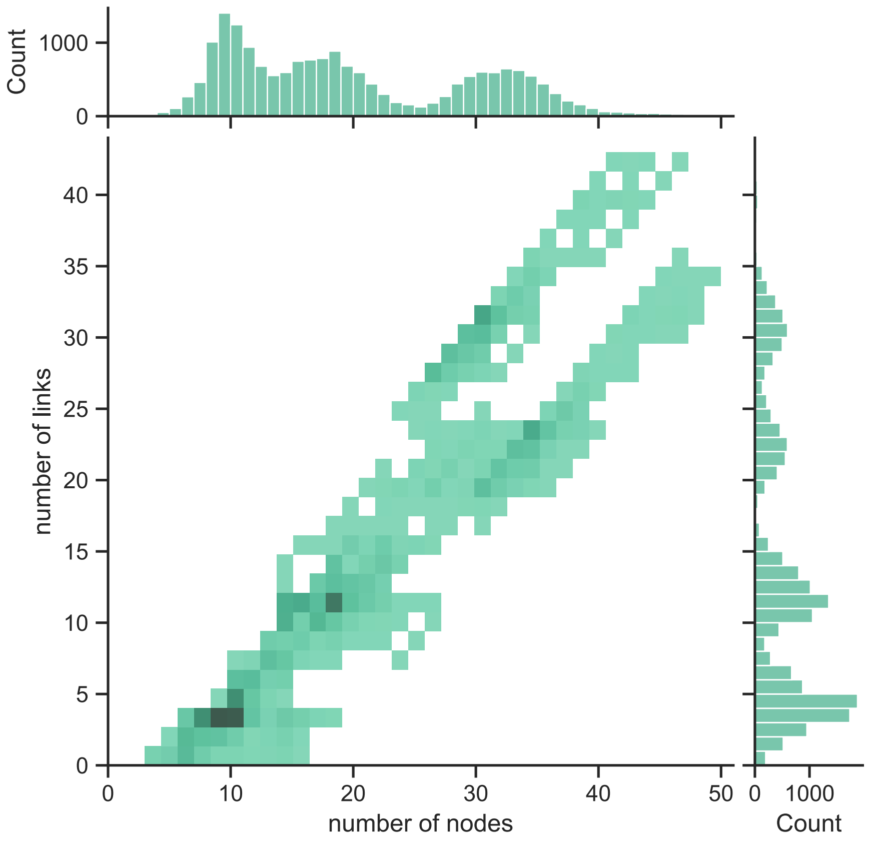
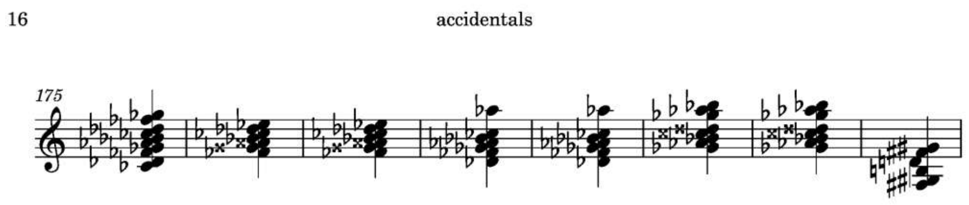


Figure 3.3: Musigraph MuNGs repartition by number of nodes and links. In the graphs, the number of edges and nodes are highly related. The diagram also shows the number of scores per number of nodes or edges is very uneven.

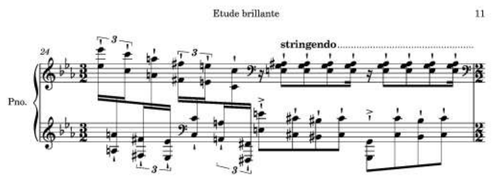
3.1.3 DoReMi

DoReMi is the most recent dataset, published in 2021 by Shatri et al. [38]. It contains a mix of real-world and synthetic scores. These synthetic scores were initially intended to test music notation software. This very rich dataset comes with different encodings, including one with MuNGs. For this version, each score corresponds to a staff or a system (group of staff read simultaneously). DoReMi counts 5'218 scores, 466'689 nodes and

274'663 relations. It uses 71 labels and 31 relation types. Details about these categories and link types are provided in Annexes A (Tables A.6 and A.5). The scores are all typeset and their structural complexity varies from monophonic to pianoform. Figure 3.4 illustrates a synthetic score and a Pianoform score from this dataset.



(a) Synthetic score extracted from DoReMi



(b) The TU Wien Informatics logo at half the text width.

Figure 3.4: Pianoform score extracted from DoReMi

.

3.1.4 Brief Comparison of the Datasets

Table 3.1 gives an overview of the dataset's main characteristics. To deal with some problems induced by the normalisation of the score, we created a version of DoReMi and Muscima++ with their scores divided by measure. This process, described in detail in Section 3.2.3 induces some data loss. Therefore, those two variations are listed with the addition measure_cut in Table 3.1.

Table 3.1: Comparison of the datasets

datasets	Number of scores	Number of primitives	Number of relations	Number of labels	Number of relation types	Type of score	Average node degree of the MuNGs
Musigraph	18921	370404	253623	18	11	Measure	1.37
Muscima++	140	102914	144386	115	217	Page	2.80
Muscima _measure_cut	2820	98095	116474	110	197	Measure	2.37
DoReMi	5218	466689	274663	71	31	Staff or system	1.18
DoReMi _measure_cut	11226	283529	169156	68	28	Measure	1.19

3.2 Dataset Encoding

This section addresses the primary challenges in creating datasets for Optical Music Recognition with music notation graphs.

3.2.1 Set of Classes

The previous sections show the lack of consensus in labelling music primitive types. We define the *set of classes* (or *class set*) of a dataset as the set of labels used to encode the types of musical primitives (notehead, tie, beam. . .). All the datasets don't use the same set of classes.

The class sets might differ in the primitives they can handle. Some primitives might not have a defined class in a class set. A good example is the decomposition of flags. Flags are primitives that tak part in the definition of the notes' duration. For the following discussion, we assume no other primitive than flags affect the note duration and the time signature is 4/4 (see the chapter "How to Read Music"). 3.5 illustrates how different flags can look like on a typeset score. The figure's leftmost note (1) has no flag, and its duration is one beat. The next note (2) has a single (red) flag, indicating that its duration should be halved compared to a note without a flag. Thus, this note's duration is half a beat. The third note, with a 'double' green flag, has a duration of a quarter of a beat, half the duration of the note with one less flag. Flags function in this manner, halving the note duration every time another flag segment is added. Traditionally multi-flags are considered atomic musical primitives rather than accumulations of single flags. In English, their naming is based on the fraction of a whole note their note occupies. For example, the red flag of Figure 3.5 is called an eighth flag because eight notes with this flag last as long as a whole note. This naming convention is used in the three datasets presented earlier in this chapter. There is no theoretical limit for the number of flag segments, although the readability of a score decreases with the number of flags and in practice, notes with more than a 64$^{\text{th}}$ flag are extremely rare. SMuFL defines flag labels until the 1024$^{\text{th}}$ fraction

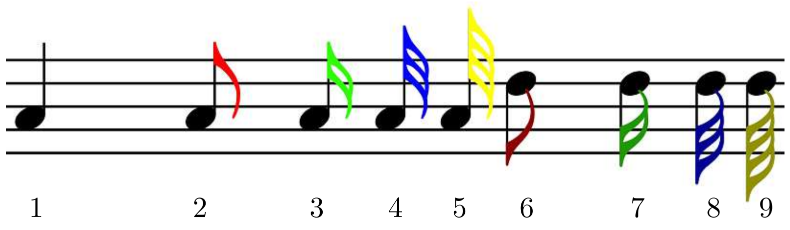


Figure 3.5: Coloured flags. Up flags point upwards, like in notes 2 to 5, and down flags point downwards, like those from 6 to 9.

An OMR system must distinguish between different flag types because they determine the note's duration. However, with this naming convention, a theoretically infinite number of labels would be required. As a result, some datasets may limit the notation they can support. The three datasets differentiate 8th from 16th flags, but only DoReMi has labels for encoding 32nd flags. The definition of the set of classes might limit an OMR system. A more flexible solution to handle any flag with a limited number of labels would consider each 'single flag segment' of a 'flag' individually and have a relation between each segment and the corresponding notehead. An OMR system could reconstruct the correct duration by counting the number of relations to flags. But none of the listed datasets works like this.

A class set can be described by its vocabulary. The vocabulary corresponds to the terminology used to designate a primitive. For instance, a completely black notehead is labelled as 'noteheadFull' in Muscima++, 'notehead-full' in Musigraph, and 'noteheadBlack' in DoReMi. Despite the different names, they refer to the exact same primitive in all three datasets. The vocabulary might depend on personal preferences and naming conventions used during dataset creation.

A class set can also be described by its granularity. The granularity of a class set defines the precision of the class set. One class of a class set can be decomposed into multiple classes in another class set. A good example is again the decomposition of flags. Muscima++ and DoReMi differentiate 'up' from 'down' flags. Even though this distinction doesn't impact the duration of a note it can have a different semantic. The direction of a flag can depend on the position within the staff to visually ease reading or indicate which voice is supposed to play the note in a polyphonic score. This distinction doesn't exist in Musigraph where a flag8thDown or a flag8thUp correspond to a 8th_flag. Flags are one type of primitive, and the question of how specific the class should be also arises for other primitive types like noteheads or accidentals.

Some initiatives try to overcome this lack of consensus in the class sets: the Standard Music Font Layout (SMuFL) [39] [40] defines codepoints as Unicode associating classes to glyphs. The names of the classes correspond to a standardised vocabulary. SMuFL defines around 2600 classes that frame a very fine granularity and recommends the use of several hundred glyphs. Initially developed for the scorewriter Dorico [41], SMuFL is now adopted by other music notation software such as MuseScore [10] and Finale. Despite its widespread use, SMuFL has not achieved universal adoption. For instance, LilyPond [11], which is a popular scorewriter, employs its own granularity. Among the three datasets, only DoReMi implements the SMuFL.

3.2.2 Music Notation Graph

The notion of "Music Notation Graph" has been used in several research [19] [42] [18], but has no commonly accepted definition. The shared understanding is that music primitives form the nodes, and edges model some interaction between those primitives. These interactions are implicit in CWMN, and their definition is the barrier to a standard definition of MuNG. Figure 3.6 illustrates a MuNG on a simple monophonic measure.

While MuNGs can exist on their own, they are typically used within the context of OMR systems. Consequently, the definitions of their links are often tailored to the specific

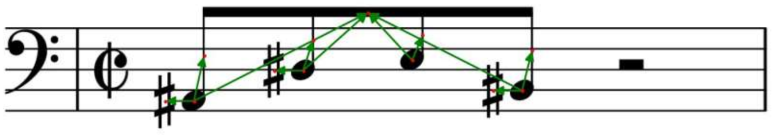


Figure 3.6: Representation of a Music Notation Graph

objectives of these systems. Some graphs utilise a rich set of interactions; for example, Pacha et al. [18] differentiate between Precedence and Syntactic edges. Precedence edges establish the temporal order of the symbols, while Syntactic edges connect the primitives that constitute a musical note. Precedence edges are unambiguous, and the temporal order of the primitives is trivial in most scores. Therefore, many MuNGs omit precedence edges. In the datasets used for implementation, only syntactic edges are included.

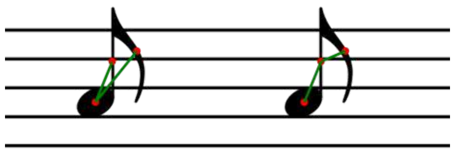


Figure 3.7: Illustration of different MuNGs on simple eighth notes

Unlike precedence edges, defining which syntactic edge should be constructed is not trivial. The relations between the primitives that constitute a note can vary from one dataset to another. For example, one could create direct links between noteheads and flags, while another would rather use a link between the flags and the stems and then a link between the stem and the notehead. Figure 3.7 illustrates these two ways of building MuNGs. Both could be used in an OMR system as they represent the semantics of the music notation. In this example, they also contain the same information. MuNGs might also differ by the information they encode: Muscima++ includes the relation between the noteheads and the staff, while the other datasets don't. One could probably claim this information is not needed in an OMR system but it might also help when multiple staff are used in a score. Other encoding choices are more questionable, e.g. DoReMi does not encode the link between accidentals and noteheads. As these links are crucial to reading the music and determining the pitch of a note, the definition of such MuNGs might not be sufficient to recover the full semantics of the music or require additional interpretation.

Another essential characteristic of the MuNGs is whether the graph is directed or undirected. For syntactic edges, whether a note's compound interacts bilaterally or unilaterally is a matter of perception, but this concept might impact the behaviour of an OMR system (see Section 4.5.1).

3.2.3 Dividing Scores by Measure

Cutting the scores by measures is a necessary preprocessing step to align the data format with the requirements of Musigraph, which operates on individual measures. Having all the datasets with the same format allows for training models on multiple datasets. This section presents the method used to divide the scores by measure and explains how this process may induce loss in the dataset.

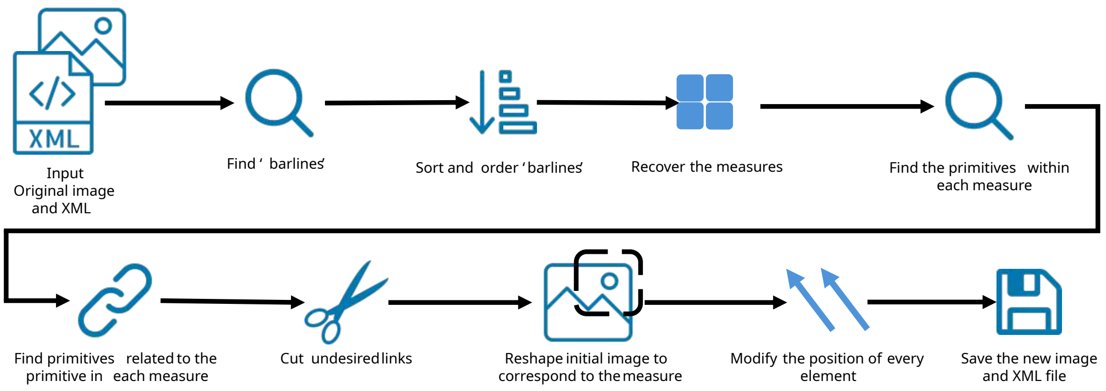

XML
Input
Original image
and XML
Find ' barlines'
Sort and order 'barlines'
Recover the measures
Find the primitives within
each measure
Find primitives related to the
primitive in each measure
Cut undesired links
Reshape initial image to
correspond to the measure
Modify the position of every
element
Save the new image
and XML file

Figure 3.8: Pipeline for dividing scores by measures

Figure 3.8 shows the pipeline used to divide the scores. The goal is to obtain the MuNG and cropped image for each measure. The measures are recovered using the 'barlines', 'barlineHeavy', or any other elements that delimit a measure. MuNGs don't include links between the two barlines of a measure[1], but once ordered with their positions, it is possible to define which barlines delimit a measure. Each measure contains a set of primitives located within an area we call the Mesasure's Surface (MS). Horizontally, the MSs are delimited by the barlines. The vertical limits are more complex to define. Theoretically, a measure has no upper or lower limit, and notes with high or low pitches might land outside the staff and receive ledger lines as visual cues. Technically, most primitives land on the staff but for scores with multiple staffs, it might not always be obvious to which staff a primitive belongs to. Two strategies are used to process the two datasets. For DoReMi, the measure's upper limit is set to an extremely high value because scores always contain a single staff or system. For Musicma++, we determined that increasing the height of the rectangles formed by the barlines and their heights by 25% was a fair compromise to define the MS.

Once the MSs are defined, a list of primitives that land on each measure's MS is defined. This list is called Primitives on the Measure's Surface (PMS). Then, for each measure, a second list of primitives linked to the primitives in PMS is created. The primitives common to the two lists are removed from the second list called Primitives Out of the Measure's Surface (POMS). POMS contain primitives from other measures or primitives

[1]The first barlline of a staff is often omitted and the first measure is often only delimited by its right barline. For simplicity, these measures are ignored here.

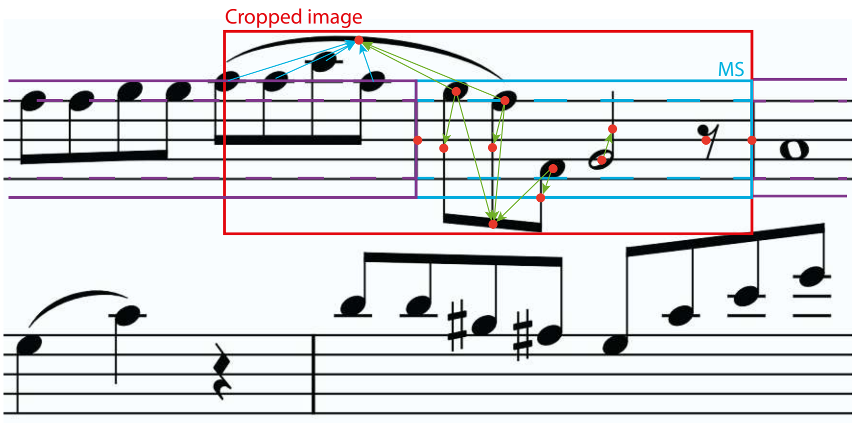


Figure 3.9: Construction of a measure. The final measure is delimited by the red rectangle and contains the primitives marked with a red dot. The green edges are included in the final MuNGs, and the blue ones are cut.

that did not land in any SM. Figure 3.9 illustrates the objects used to construct a measure. In this example, the POMS contains the slur and the beam with the red dots. The other primitives marked with a red dot and landing in the blue MS belong to the PMS. The links between the elements in POMS and elements not in PMS are cut. These links are blue in the example. The MuNG of a measure is then built with the primitives of the two lists. Staff are excluded from the primitives for simplicity: in Muscima++, they are linked to every notehead. The POMS of most measures would contain all the noteheads of the score and include them in every MuNG. Removing these links explains the drop in node degree between Muscima-pp and Musicima_measure_cut in Table 3.1. For each measure, the left-most, right-most, top-most, and bottom-most primitive's bounding boxes define the boundaries used to crop the input image. The position of the primitives is mapped to correspond to this new image. Finally, the MuNGs and the cropped image are saved.

This process has multiple advantages. It ensures that the correctly located primitives' relations are preserved in the ground truth. Every relation in the produced scores exists in the original ground truth; no edge can be arbitrarily added. Some primitives can belong to multiple measures, which is crucial for dynamics or slurs. Saving the ID of these duplicate primitives enables reconstructing the initial MuNGs, making this division reversible. However, this process has many drawbacks. First, many scores do not include barlines to define the beginning of the first measure of staff, and the process omits many measures. Then, the definition of the MS is imperfect. Some primitives might be excluded or wrongly attached to multiple measures, creating incomplete MuNGs or scores that do not correspond to a measure.

The number of errors in these new datasets is unknown, but no problem was discovered when using the dataset. We are convinced that other systems could solve this problem in a much more accurate way and would preserve a greater share of the initial dataset. Nevertheless, the new datasets seem sufficient for evaluating GNNs in the context of semantic reconstruction for OMR.

CHAPTER 4

Semantic Reconstruction of Music Notation With GNNs

4.1 Pipeline Overview and Practical Considerations

Figure 4.1 illustrates the pipeline implemented to train a GNN-based model for reconstructing the semantics of the music notation. The process can be split into three main steps. First, building a feature matrix from an XML input. Secondly, building the Candidate Graph (CG) that serves as model input. This is achieved by adding an adjacency matrix to the feature matrix. At the same time, the ground true graph is constructed using the same feature matrix and the relations listed in the ground truth. The last step involves training the GNN-based model.

GNNs require graph-structured input. Unfortunately, musical object detectors output a list of nodes with their label and position. In other words, the pipeline input can be turned into a node feature matrix but ultimately lacks the accompanying adjacency matrix. An early step in using GNNs is to transform the grid-like output of the musical object detection into a graph-like input, i.e. to recover a feature matrix and build an adjacency matrix from scratch.

To build the adjacency matrix, the first intuition would consider every node to be connected. Veličkovič et al. [24] state that this approach is valuable for small graphs as it exploits the full potential of a GNN. Node information is shared across all nodes, and the model can learn from it. Under this 'fully connected graph' assumption, it can be shown that graph attention-based GNNs are equivalent to transformers [33]. However, this approach doesn't scale well and hasn't been extensively tested in this work. The opposite intuition would consider every node to be disconnected. GNNs update node representation based on their neighbourhood; in this case, a node representation would be updated only based on its own features. Any GNNs would be equivalent to the Deep Sets

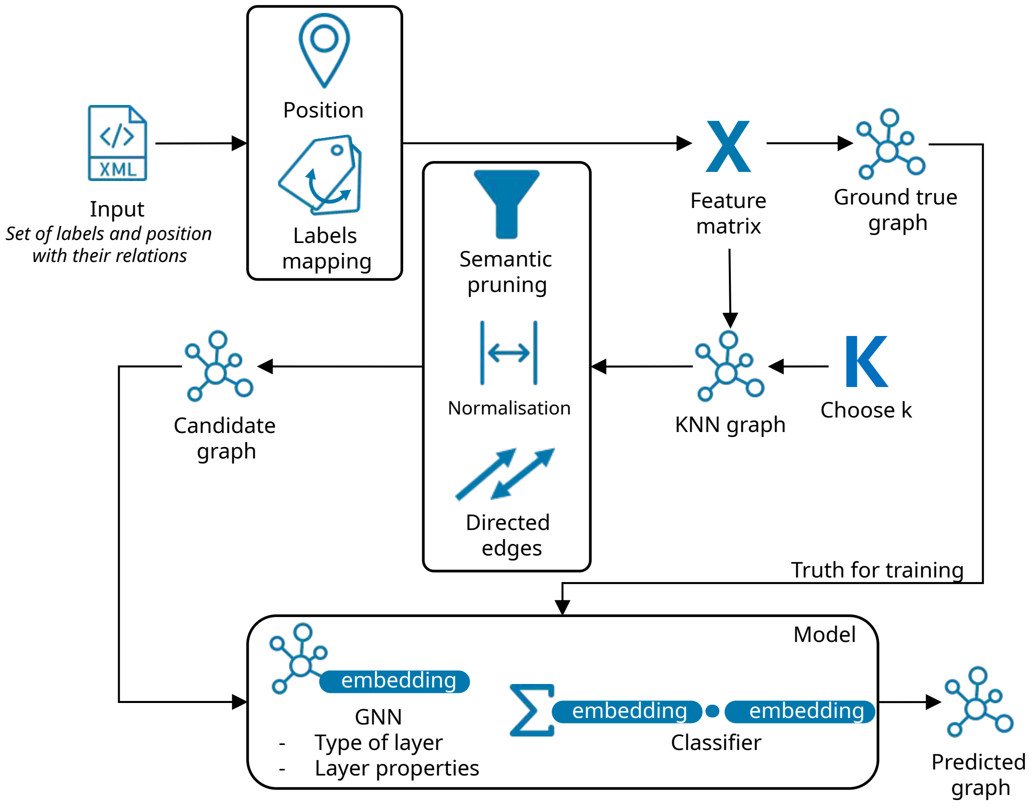


Figure 4.1: Training Pipeline

model [43], and no benefits of the GNNs functioning could be exploited. This approach has, therefore, been excluded. The last possibility is to use rules or to infer a graph from the feature matrix. Inferring such a graph is an emerging area called Latent Graph Inferring. The key idea of this approach is to embed the latent features of the nodes into high-dimensional spaces and predict the graph edges from the proximity of the nodes in this hyperspace. Notable work is the Differentiable Graph Module (DGM) [44], which introduced this idea and marked a significant breakthrough in the usual benchmarks. Recently, Ocariz Borde et al. [45] improved this framework by leveraging planes with non-constant curvature in the hyperspace instead of the Euclidean planes used to build the DGM hyperspace. Such an approach could be combined with a GNN but could also replace the whole pipeline and offer an end-to-end alternative. However, this solution's lack of interpretability leads to considering a rule-based approach to build the CG. We define the *positive edges* as edges a candidate graph and a ground truth graph have in common. Similarly, the *negative edges* are the edges of a candidate graph that are not in its ground truth. Note that all the edges of the ground truth might not be included in the CG. Having both types of edges in the CG is crucial to enable the model to learn when to predict edges. Using the observation that related music primitives are spatially close to one another and the work of Baro et al. [19], employing a K-Nearest Neighbors

(KNN) approach appears to be a promising solution. A KNN graph constructs edges between a node and its k nearest neighbours. Having such a graph should include both negative and positive edges. The optimal value for k and strategies to enhance this graph are discussed in Section 4.4.1.

A link prediction task on a graph aims to predict an edge feature. To perform such a prediction with GNNs, a classifier uses the representations of the two endpoints of a potential edge. This edge is called a candidate edge (CE). Usually, the classifier predicts the existence of a set of CE, and every other edge that is not included in this set, is considered inexistent. Defining a set of CE that includes every edge of the ground truth is necessary to reconstruct a graph perfectly. This set of candidate edges is, in practice, independent from the edges of the CG. To define the set of candidate edges, the first intuition is to include every possible edge of the graph to predict. This method ensures that with a perfect model, the ground truth can be recovered, but this method is computationally expensive as the number of edges in a fully connected graph is given by $f(n) = \frac{n(n-1)}{2}$, with n the number of nodes in the graph and undirected edges. To reduce the number of predictions, a restricted set of CE can be defined with a heuristical approximation of the desired graph that includes at least all the ground truth edges. Using the observation that related music primitives are spatially close to one another, the KNN approach is a natural candidate. Considering that the CG and the set of candidate edges are obtained similarly, the candidate edges could be defined by the edges in the candidate graph. This selection of CE would be very efficient since the edges are already available from the GNN training. As the links already exist, the task can be considered as link classification instead of link prediction, but this distinction is rarely made in the literature. Using such a method requires some practical considerations:

The set of CE must contain the set of ground-true edges; this is not a requirement for the candidate graph, which can be built arbitrarily. To use the edges of the candidate graph as CE, the candidate graph must include every ground-true edge. We define Inclusive Candidate Graph (ICG) as candidate graphs that include their corresponding ground truth. Section 4.4.1 describes how to assert candidate graphs are ICG.

Another factor to consider is the versatility of GNNs, which can accept any graph as input. In our study, various pipelines can interpret this flexibility and construct graphs differently during the training and testing phases. Building smaller graphs for training the GNN could improve the training time while capturing most of the information needed to reconstruct the semantics of the music notation. To evaluate this model, another pipeline could then construct ICGs, which might be larger than the training graphs. This is discussed in Section 4.4.1.

4.2 Metrics

The following sections detail the parameters and methods used to construct the training pipeline. To assess the impact of certain parameters, experimental results are presented throughout these sections. As some of these metrics are unusual, they are presented upstream in this section.

4.2.1 The Problem with OMR

Chapter 3.1 shows the need for more consensus on building datasets for OMR that contain MuNGs. Despite a few standards, the definition of MuNGs edges and class sets are specific to each dataset. These encoding imperfections are also reflected in the evaluation standards: There is no standard metric or method to rigorously describe and evaluate the output of the music semantic reconstruction stage, and specifically, no metrics for MuNGs. This known problem has already been discussed in the literature [46] [16] [47].

Transitioning from the output of the music semantic reconstruction stage to the final output is claimed to be straightforward. This task hasn't been done in this work, but we claim that a perfect semantic reconstruction system should be able to recover all the musical information of a score. This includes recovering advanced implicit information like omitted primitives; a running example of such primitives concerns the triples. These notes visually look like three eight notes linked by a beam with a '3' on top, but the '3' is often abusively omitted after the first triple. Figure 4.2 shows the score of the beginning of the Moonlight Sonata from Beethoven, it illustrates this omission of '3' above the triples. A perfect semantic reconstruction system should understand such omission. With this claim, the last stage of the OMR pipeline corresponds to the encoding of only (and, if possible, all) the information obtained at the music semantic reconstruction stage. Therefore, the evaluation of the semantic reconstruction stage and the evaluation of the final output of an OMR system can be indiscriminately discussed.

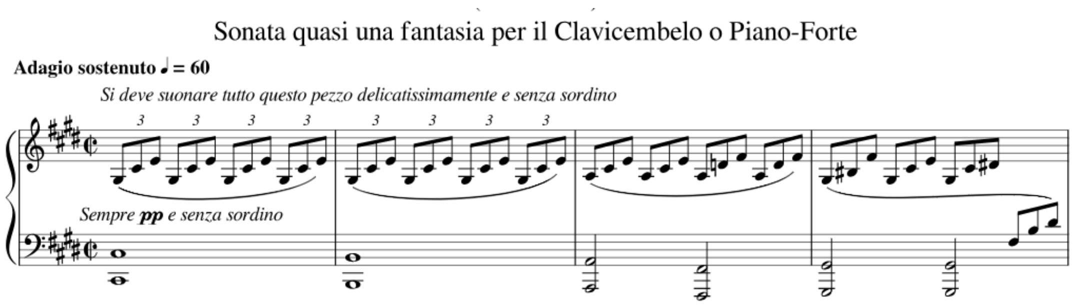

Sonata quasi una fantasia per il Clavicembelo o Piano-Forte
Adagio sostenuto ♩ = 60
Si deve suonare tutto questo pezzo delicatissimamente e senza sordino
3
3
3
3
3
3
3
3
Sempre pp e senza sordino

Figure 4.2: First measures of the Moonlight Sonata from L. van Beethoven. The two first measures contain triples with the '3' on top. On the following measures, the triplets are missing their '3'.

In [46], Hajič et al. distinguish between two types of evaluation methods: *extrinsic* and *intrinsic*. Extrinsic methods evaluate systems in application contexts. For example, how well does an OMR system address transcription? Intrinsic evaluations don't require any context and would instead refer to the share of information an OMR system manages to recover. As the information contained in a music score is limited, there is no reason why such a metric cannot exist. The authors also state that there is no automatic method for OMR that is (1) rigorously described and evaluated, (2) has a public implementation, and (3) gives meaningful results. In other words, he claims that no intrinsic evaluation method has been found for OMR yet. Such metrics would greatly benefit the field and enable a fair comparison of OMR models. However, even in 2024, no intrinsic method has been found, and comparing models often requires expensive user studies [47].

Despite their drawbacks, some standard metrics remain valuable for evaluating OMR systems. For our specific case of MuNG, we can define two types of metrics: those that rely on the task's binary classification aspect and those that rely on its graph aspect.

Binary Classification Metrics

The task of music semantic reconstruction can be described as a binary classification task where each link of a graph would be classified as existing or not. With this formulation, a model can be evaluated with standard metrics of classification tasks. We denote:

$$\begin{array}{lcc} & \text{Predicted Negative} & \text{Predicted Positive} \\ \text{Actual Negative} & TN & FP \\ \text{Actual Positive} & FN & TP \end{array} \tag{4.1}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.4}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{4.5}$$

These metrics have important implications. They enable a fair comparison of models' performance only under certain conditions. The model inputs must have similar input graphs and a similar proportion of edges in the ground truth and the input graphs. In our case, the number of neighbours used to create the KNN graph and the edge filtering could make these metrics meaningless. Let's take two examples to put that into perspective.

Example 1: Restrictive input graph

> Let's imagine we set the number of neighbours of the KNN too low, and the generated CG doesn't cover every edge of the truth (see Section 4.4.1) for details on the graph behaviour under the influence of k). Let's also imagine that our model has 100% accuracy, and that every metric described above is maximised.
>
> Although this model's output is 100% accurate, it would not recover all the information needed to generate the desired output of the global OMR system because some edges of the ground truth will still be missing.

Example 2: Over-Saturated input graph

> Assume two CGs, A and B, are generated for the same score. B is a simple KNN graph, while some negative edges in A are pruned. Let's now imagine a model that generates the same output (with some mistake) for the links A and B have in common. Let's also assume the model correctly cut the extra edges B has compared to A.
>
> In such a scenario, the accuracy (or any other metric listed above) would be higher for B than for A, but the quality of the output would strictly be the same from an OMR perspective. This problem occurs the same way if B has many negative edges that can easily be cut.

These issues don't make the binary metrics useless; they reflect how well a model learns rather than how interesting its output is. They remain precious tools as long as we compare models on the same inputs. Some drawbacks of these metrics vanish if we consider the output not as a binary classification but as a graph.

4.2.2 Graph-based Comparison Metrics

The ground truth and the output of the pipeline can be seen as MuNGs, which are graphs. From this angle, we can consider evaluating our model with graph-based evaluation criteria. Graphs can be complex structures, but a considerable research effort has been made to define comparison metrics for these structures [48]. MuNG are simple graphs, and because we favour easy-to-interpret metrics, Graph Edit Distance (GED) seems to be the most suitable metric. In *A survey of graph edit distance*, Gao et al. [49] state that a finite sequence of graph edit operations can transform a graph into any another one. They define the GED as the least-cost edit operation sequence. Having two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the GED can be defined as follows:

$$\mathrm{GED}(G_1, G_2) = \min_{(o_1, o_2, \ldots, o_k)} \sum_{i=1}^{k} c(o_i) \tag{4.6}$$

With $o_1, \ldots, o_k$ a set of operations that transforms G_1 into $G2$ and $c(o_i)$ the cost of the operation i.

For our needs, we can make two assumptions to simplify the GED:

- **A1**: V1 and V2 are the same ensembles: We consider the nodes (music primitives) to be perfectly detected; hence, our ground truth and our prediction always share the same set of nodes.
- **A2**: The editing operations are limited to (1) adding and (2) cutting an edge. The two operations have the same cost. A2 enables to transform any graph $G_1 = (V, E_1)$ into $G_2 = (V, E_2)$.

Under these assumptions, we can simplify the GED as:

$$GED(G_1, G_2) = \min(|o_1, o_2, \ldots, o_k|) \tag{4.7}$$

With $|.|$ the length of the sequence.

$$GED(G_1, G_2) = |E_1 \cup E_2| - |E_1 \cap E_2| \tag{4.8}$$

This GED breaks some limitations the binary-based metrics have. The value of the GED cannot be manipulated by the size of the input graph, and under the two assumptions A1 and A2, for the same ground truth, a lower GED will always correspond to a better output. This metric is very meaningful for comparing the outputs of different models for the same score regardless of the model inputs.

The drawback of the GED comes when we aim to interpret its value for different types and sizes of scores. Let's imagine we want to evaluate a model on various types of scores, e.g. monophonic vs polyphonic. We assume the polyphonic scores will have much bigger ground truth graphs than the graphs corresponding to the monophonic scores. It could be meaningless to compare the actual GED values obtained on the scores because, for the same value, a much greater share of the edges would be recovered for the polyphonic scores.

This drawback probably led Baro et al. to instead rely on the Music Error Rate in [19], which they define as follows:

$$MER = \frac{I + R + S}{T} \tag{4.9}$$

where I, R, and S are the number of insertions, deletions, and substitutions to obtain the ground truth sequence. T is the number of edges in the ground graph.

This metric is designed to evaluate the second and third stages of the OMR simultaneously: it considers substitution, which corresponds to relabeling music primitive. The two other operations, insertion and deletion, can be applied to the edges as well as to the nodes. Our task is limited to the third stage of the pipeline, but we could use MER as it is defined. For an easier interpretation of the result and under assumptions A1 and A2, we can simplify the MER and redefine it as follows:

$$MER = \frac{GED}{T} \tag{4.10}$$

With this form, MER appears very close to the GED but solves, to a certain extent, the problem we stressed for this metric. Since we divide the GED by the length of the ground truth, we can compare the MER value for scores with different lengths. However, the interpretation of the MER value remains very tricky: As the inverse function is involved, calculating the MER for each measure separately and then averaging those MERs gives a result that isn't directly comparable to the MER calculated for an entire graph. It is important to be aware of this when comparing or combining MER values. The other limitation of the MER is the choice of dividing the GED by the length. The length of the ground truth is not an obvious candidate. One could claim that the number of primitives could be a more natural choice regarding interpretation. In fact, we aim to balance the GED by the score's complexity and a better metric would use an objective measure of the music notation complexity. Such a metric does not exist (yet). Rigorously assessing the complexity of the music notation seems to be a research gap and no paper addresses it. However, measuring the difficulty of playing a certain piece of music based on its score benefits from a lot of research [50] [51]. These papers could serve as a basis for creating a metric for the music notation complexity, but this task is out of the scope of this thesis.

Are GED or MER the Missing Metrics of the OMR Field?

According to Hajič [46], the ideal method to evaluate an OMR output needs to (1) be rigorously described and evaluated, (2) have a public implementation, and (3) give meaningful results. With a clear definition of MuNGs and a defined class set, only one ground truth graph can be constructed for a score. The GED or MER can also be clearly established by determining the different edit operations. GED or MER would then correspond to the share of the music information an OMR system managed to recover. The two first characteristics defined by Hajič are fulfilled in this condition. The last characteristic concerns the interpretation of the metric. The use of 'meaningful' in the description of the ideal metric lacks clarity. We demonstrated that interpreting these metrics is not always straightforward, but one could easily claim they are meaningful.

4.2.3 Test Procedure

Experimental results are presented throughout the thesis to assess the impact of certain parameters. The testing procedure to obtain these results consists of running a baseline configuration altered by one (or two) change(s) at a time. All the test results presented in this work have been obtained using a CPU to ensure reproducibility.

By default, a configuration is tested on the dataset Musigraph. The candidate graphs are based on KNN graphs constructed using 14 neighbours. Their edges are considered directed (The information can only flow along the direction of the edges), and the scores are normalised. The model is a GNN composed of 3 graphSAGE layers, having intermediate representations of size 2048. The final layer outputs a vector of size 1x1024 for each node. Between each layer is a ReLU activation function and the final output is normalised. The model learns over 100 epochs. The learning rate starts from 3 and is updated during the training by a scheduler that reduces the learning rate by a factor of 0.5 when the validation loss stops improving for a patience of 10 epochs. The imbalance of positive to negative edges is considered with a weight computed on the training set before the first epoch.

The semantic pruning is a process that filters the edges of a CG that could never exist in a ground truth based on their endpoints' labels. This process is described in detail in Section 4.4.2. Pruning the graph might modify the impact of other parameters: On pruned graphs, a parameter can improve the pipeline but make it worse when the graphs are not pruned. By default, the tests are run first without pruning and then with pruned graphs. If the best configuration of the test doesn't change with the pruning parameter, only the first test is saved. The training pipeline is shaped by other specific parameters presented throughout the thesis, their default values are available in Annexe B.

The test output contains a confusion matrix built on the test set as well as an average MER, GED, and accuracy. In addition, an analysis of the output includes the accuracy obtained for each link type, e.g. the accuracy of the model on the relations between noteheads and stem. An option enables a visual display of the predicted graphs for a more human-friendly output. The original image is superposed with the graph's true positive, false negative, or false positive edges. As true negative edges are usually numerous and not very insightful, they are not visualised. Figure 4.3 shows an example of such graphs. The colour also indicates the error type.

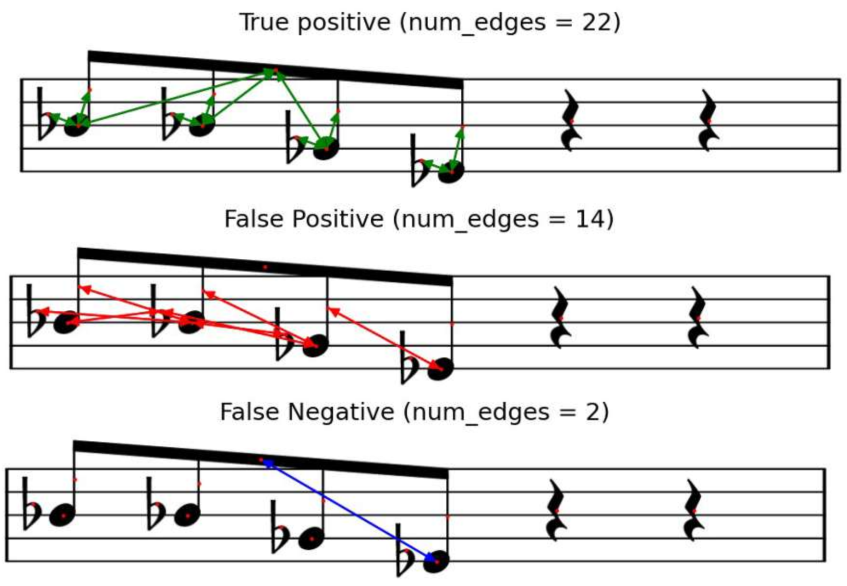

True positive (num_edges = 22)
False Positive (num_edges = 14)
False Negative (num_edges = 2)

Figure 4.3: Example of graph visualisation

4.3 Data Preparation: from XML to feature matrix

This section discusses the first step of the training pipeline. It explains how to obtain the feature matrix of the ground truth and candidate graphs from the input. This matrix is composed of the feature vector of the graph nodes and is referred to as X in the GNNs literature (see 2.2). After a music object detector, the feature vector of a node can be defined using the primitive's label, sizes and position.

4.3.1 Encoding the Labels

We decided to one hot encode the node's label in the feature vectors. However, the set of labels to use is not straightforward. This work aims to build models that deal with scores from different datasets. To handle the different sets of labels of the dataset (see Section 3), the first step of the pipeline maps each label into a common set of labels.

For some experiments, the set of classes employed in the datasets are used to perform the one-hot encoding. These class set are called *all_dataset_name_labels*. They can only be used with their corresponding datasets. Conversely, *mono_label* is a set of classes that maps every label to the same 'primitive' label.

Defining other sets of classes involves trade-offs and decisions. We can decide the granularity of the set: E.g., to label flags, one approach could include using multiple specific labels, such as '*8th_flag_up*', '*8th_flag_down*', '*16th_flag_up*', *16th_flag_down*',.... Alternatively, a single, more generalised 'flag' label could be used to encompass all variations. This echoes the problem described in Section 3.2.1, but the definition of granularity here doesn't have the same implication as the granularity of the dataset's class set. Each node (or primitive) is associated with an ID, and it is always possible to return to its original label (the one given by the object detector). For example, every accidental can be labelled 'accidental' while having a different influence on the noteheads (sharps increase the pitch by half a ton while flat do the opposite). After the model predictions, using the ID of the node, it is possible to recover the original label and encode the final representation of the score in consideration. It is, therefore, always better to have the most detailed granularity in the object detector, as it can be simplified at this step of the pipeline. To build a proper class set for mapping multiple datasets, the least specific granularity of the datasets' class set is the limiting element: In Musigraph, the distinction between empty and half-empty noteheads doesn't exist, while Musicma++ and DoReMi make it. If the set of classes used for mapping the object detector labels distinguishes between these two types of noteheads, mapping Musigraph's noteheads to their new labels would be imperfect; it would mix the two kinds of primitives under one of the two more specific labels.

To define the set of classes, a vocabulary must be defined too. Since the labels are one-hot-encoded, the models never get the actual name given to the primitive. The vocabulary is only used to order the features when the one-hot encoding is performed.

Table 4.1: Mapping to 10_labels class set

Musigraph	Muscima++	DoReMi	10_labels
stem	stem	stem	stem
notehead-full, notehead-empty	noteheadFull, noteheadHalf, noteheadFullSmall, noteheadWhole	noteheadBlack, noteheadHalf, noteheadWhole	notehead
beam	beam	beam	beam
(not existing)	augmentationDot	augmentationDot	augmentationDot
sharp, flat, natural	accidentalSharp, accidentalFlat, accidentalNatural, accidentalDoubleSharp	accidentalSharp accidentalFlat, accidentalNatural accidentalDoubleFlat accidentalQuarterToneSharpStein accidentalQuarterToneFlatStein accidentalDoubleSharp accidentalThreeQuarterTonesSharpStein	accidental
16th_flag, 8th_flag	flag16thUp, flag16thDown, flag8thDown, flag8thUp	flag16thUp, flag16thDown, flag8thDown, flag8thUp, flag32ndUp, flag32ndDown	Flag
(not existing)	tie	tie	tie
(not existing)	legerLine	(not existing)	legerLine
(not existing)	dynamicCrescendoHairpin, dynamicDiminuendoHairpin slur	dynamicForte, dynamicPiano, dynamicFFF, dynamicPPP, dynamicFF, dynamicText, dynamicMP, dynamicFortePiano, dynamicPP, dynamicSforzato, dynamicMF, dynamicForzando, gradualDynamic, slur	others_slur_dynamics_etc
8th_rest, 16th_rest, quarter_rest, half_rest	rest8th, rest16th, restQuarter, restHalf, restHBar, restWhole	rest8th, rest16th, rest32nd, restQuarter, restHalf, restWhole	rest

The definition of the class set can also be used as a filter for the nodes that cannot be involved in any relationships. Chapter 3.1 shows that some primitives are never part of relations and can thus be ignored. For example, the barlines or the time signatures are always isolated elements. Having them would probably pollute the Candidate graphs. As for the precision of labels, the elements filtered at this step are not lost. Indeed, they can be added directly to the final representation.

Multiple class sets have been defined with a mapping to the datasets' classes. Most correspond to intuitive choices guided by the behaviour of the primitives in CWMN. The simple set of classes called *10_labels* shown in Table 4.1 maps the relationships' most critical and frequent endpoint labels to 10 standard labels. The mapping to this set of classes filters many primitives like the text-based labels of Muscima++ but covers the most important labels to reconstruct the music.

To evaluate the impact of a set of classes on model performance, we used the test described in Section 4.2.3 For this experiment, three different sets of classes are tested: *10_labels*, *full_musigraph_labels*, and *mono_label*. Table 4.2 presents the test results. The vocabulary used in the class set doesn't really matter but the granularity might impact the model. The intuition is that having a larger number of classes might let the model learn finer rules. For example, if there is a down flag, the model could learn that the notehead should be left from its stem. However, having more labels makes learning more complex for the model as it enables learning on more combinations. For this test, the class set *10_labels* performs best or equivalently on two out of three indicators, which supports our intuition. However, the pipeline configuration used to train these models is relatively simple, and all these models perform poorly. By default, the class set 10_labels is used in the test configuration.

Table 4.2: Influence of the class set on the semantic reconstruction

Metric / Class set	Accuracy (%)	MER	GED	Number of labels
mono_label	**82.43**	2.79	44.96	1
10_labels	**82.43**	**2.47**	38.61	10
full_musigraph_labels	82.07	**2.47**	**37.40**	18

4.3.2 Encoding the Size and Position

In addition to their label, music object detectors provide information about the size and position of the primitives. This information can enrich the feature vectors of the candidate graph nodes. Usually, size and position are conveyed through bounding boxes: rectangular shapes framing the music primitives' symbols. These simple objects can be defined by four coordinates or distances to a reference point. Figure 4.4 illustrates the bounding box of a G clef.

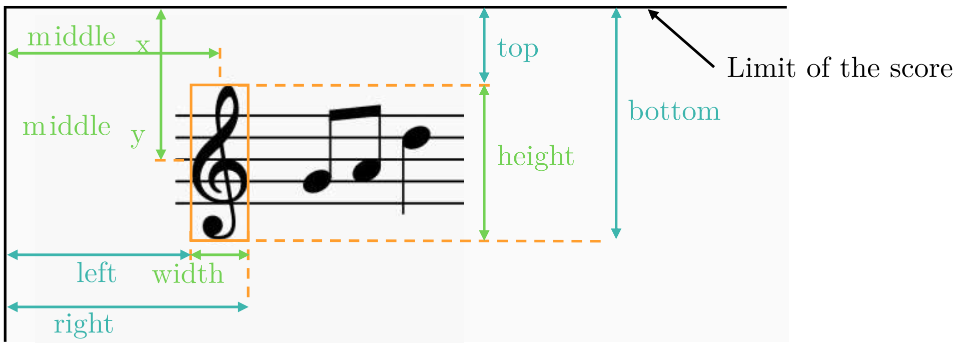


Figure 4.4: Primitive size and position encoding

To build the feature vector, we determined two sets of distances to encode size and position. One, illustrated in blue in Figure 4.4, utilises the distances between the top, bottom, left and right edges of the box to the limit of the score. Alternatively (in green), the position of the primitive is described with the centre of the bounding box ($middle_x$ and $middle_y$) and its size with the width and height of the box. The first set of distance is referred to as *bounding_box* encoding in opposition to the second set called *center_dimension* encoding. The information in the two configurations is strictly equivalent. Therefore, it is natural to think that this choice of encoding should not impact model performance in any way.

Table 4.3 presents the results of the test procedure for encoding size and position. The table also includes the results from testing with pruned graphs, where pruning involves removing certain negative edges based on their labels. This pruning process results in significantly smaller candidate graphs compared to the unpruned configuration (refer to Section 4.4.2). Although both models have access to the same information, their performance varies depending on the encoding method used. The table highlights that size and position encoding can significantly impact model performance. However, the effect of these parameters is not straightforward: in the first configuration (unpruned), the model using the *center_dimension* encoding outperforms others across all metrics, while in the second configuration (pruned), the *bounding_box* encoding yields the best results. By default, the encoding Center_dimension is used in the test configuration.

Table 4.3: Influence of the primitive size and position encoding on the semantic reconstruction

Pruning	Metric / Encoding	Accuracy (%)	GED	MER
No	Bounding_box	82.07	38.61	2.47
	Center_dimension	**85.84**	**30.56**	**2.12**
Yes	Bounding_box	**84.19**	**7.35**	**0.47**
	Center_dimension	81.26	9.07	0.53

4.4 Building the Candidate Graph: An Adjacency Matrix From Scratch

The previous section explains how a feature matrix can be built from the output of a music object detector. This section delves into constructing the remaining element of the candidate graph: the adjacency matrix. It focuses on KNN-graphs-based approaches, as explained in Section 4.1. In this section, the process is presented using Musigraph. It is the same for all datasets.

4.4.1 K-Nearest Neighbour Graph

Principle

K-Nearest Neighbour graphs are built by constructing an edge between each node of a graph and its k nearest neighbours. To construct such a graph, the centre of the music primitive's bounding boxes is used as the node position. Alternatively, the distance between the boxes could be used, but this hasn't been considered for simplicity. The distance between the nodes is obtained using the Euclidean distance expressed in Equation 4.11.

$$euclideanDistance(A, B) = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2} \tag{4.11}$$

By default, KNN graphs are simple graphs, meaning they adhere to 2 conditions: (1) they do not contain self-loops (edges connecting a node to itself), and (2) they have at most one edge between any two nodes in undirected graphs, and at most two edges (one for each direction) in directed graphs. In this work, KNN graphs are always constructed as directed graphs, but this does not imply that edges in the candidate graphs are treated as directed edges.

Choice of k

An important parameter for creating the KNN graphs is k, the number of neighbours to generate. This section discusses the impact of the number of neighbours on the candidate graphs and identifies interesting values for k in the context of music semantic reconstruction.

The script *k_neighbours_exploration.py*, whose pseudocode is described in Table 3, allows an understanding of the impact of k on the candidate graph. This script generates KNN graphs for a dataset using various values for k and extracts information such as the total size of the graphs and their proportion of positive edges. Then, for each graph, it counts the number of edges in the ground truth that are not in the KNN graph. Three configurations are tested with two class sets and the pruning or not of certain edges. Pruning the graph filters the edges that could never exist in a ground truth based on their endpoints' labels, e.g. two noteheads can't be linked in a ground truth because it has no musical sense. Section 4.4.2 gives a detailed explanation of this process. For this exploration, the graphs are generated with directed edges, such the number of links can be interpreted as the number of relationships between nodes.

Algorithm 4.1: k_neighbours_exploration algorithm

```
k_neighbours_exploration ()
begin
    configuration ← [(all_musigraph_labels, no_prune), (10_labels,
      no_prune), (10_labels, prune)]
    k_values ← [1, 2, ..., 14]
    candidateGraphs ← []
    graphSizes ← []
    proportionOfTrueEdges ← []
    countUncoveredEdges ← []
    groundTrueGraphs ← []
    foreach for k in k_values do
        foreach labels, prune in configuration do
            groundTrueGraph ← generateTrueGraphFromXML(labels)
            groundTrueGraphs.append(groundTrueGraph)
            foreach trueGraph in groundTrueGraphs do
                candidateGraph ← generateCandidateGraph(k, trueGraph)
                if prune then
                    pruneEdges(candidateGraph)
                end
                graphSizes.append(len(candidateGraph))
                numberOfTrueEdgesInTheGraph ←
                  CountCommonEdges(trueGraph, candidateGraph)
                proportionOfTrueEdges.append(numberOfTrueEdgesInTheGraph
                  / len(candidateGraph))
                countUncoveredEdges.append(len(trueGraph) -
                  numberOfTrueEdgesInTheGraph)
            end
        end
    end
    return sum(graphSizes), mean(proportionOfTrueEdges),
      mean(countUncoveredEdges)
end
```

Figure 4.5 shows the evolution of the size of the candidate graph for an increasing number of neighbours in the KNN graphs. The data are available in a tabular format in Annexe C. The number of edges increases almost linearly with the first values of k. The R^2 of the linear regression of the curves for k between 1 and 13 are superior to 0.997. The non-exact linearity is explained by the saturation of some graphs. For the score having less than k+1 nodes, the KNN graph corresponds to a fully connected graph. Increasing k doesn't change the number of edges for these scores.

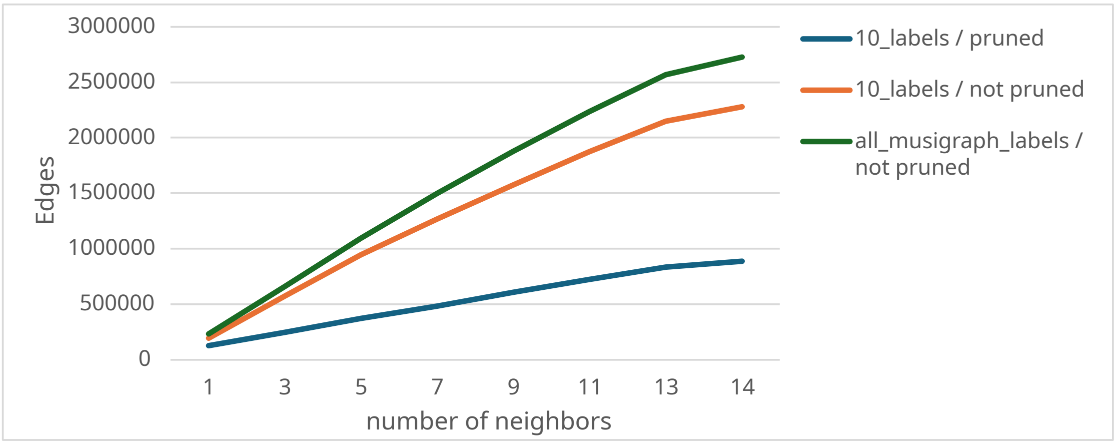


Figure 4.5: Evolution of the number of edges in the KNN by the number of neighbours

Figure 4.6 shows the proportion of positive edges in the KNN graphs for an increasing number of neighbours. This proportion quickly drops when k growth and the high proportions for smaller k values indicate primitives are likely to be linked to their closest neighbours. For the configuration '10_labels / pruned', 99.29% of the primitives are linked to their closest neighbours in the ground truth. From a machine learning perspective, this proportion corresponds to a class imbalance between positive and negative edges. This proportion is used in the loss function during the training, but it is systematically computed on the training set to prevent data leakage.

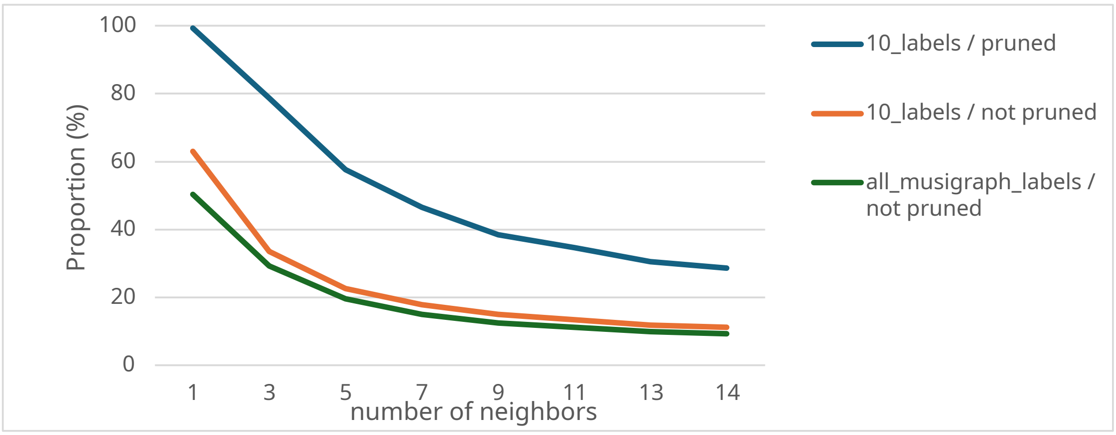


Figure 4.6: Proportion of positive to negative edges in the graph for different values of k

Figure 4.7 shows the evolution of the proportion of the ground truth edges covered by the KNN graphs. This proportion is strictly the same for configurations having the same granularity. 100% means that for each score, the ground truth graph is a subgraph of the

KNN graphs. This is achieved with k set to 14 for the class set all_musigraph_labels and k=13 for the other configuration. In a musical sense, it means that in a monophonic measure, a primitive only binds relations among its 13 or 14 closest neighbours. This statement should be taken carefully as it remains possible to write a measure with further distant related primitives, and it has been computed on Musigraph only; Musigraph contains monophonic scores, and more complex notation might require a higher value for k. The definition of the MuNG is also important to consider. Some MuNGs include links between primitives and staves, which create distant relations and require large values for k to cover all edges. The KNN graphs are used as candidate graphs, and the question raised in section 4.1 finds an answer: To ensure that the ground truth is included in the candidate graph, k must be set regarding the granularity. And any value over 14 assures this property for Musigraph.

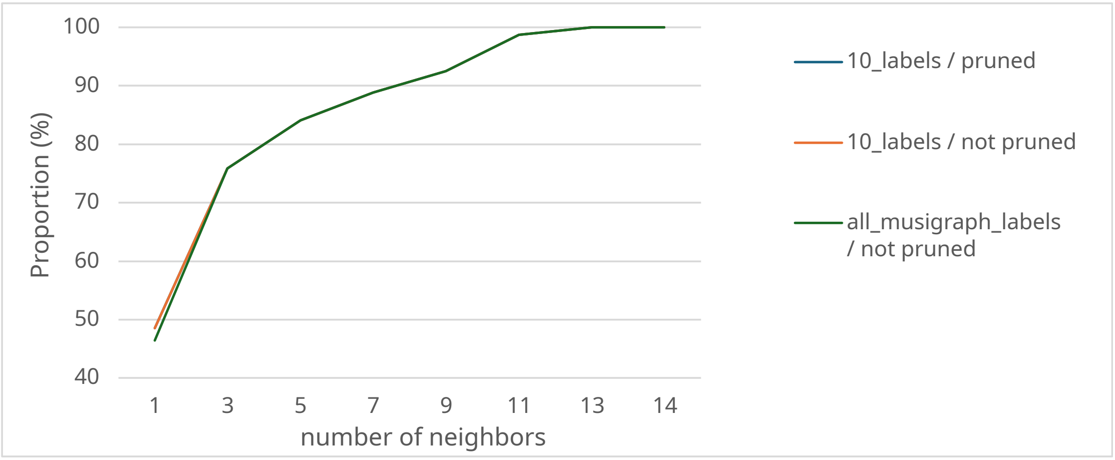


Figure 4.7: Proportion of positive edges covered by the KNN graphs by number of neighbours

Towards finer candidate graphs?

A means to get smaller Candidate Graphs while including their ground truth could be to select the number of neighbours (k) based on some metadata, such as the number of nodes in the feature matrix. Figure 4.8 shows the average share of ground true edges included in the CG by the number of nodes in the CG and the value of k. The green area corresponds to the configurations where the ground truth is included in the CG. The upper right triangle corresponds to fully connected graphs for which increasing the number of neighbours doesn't change the number of edges. The purple-to-blue area corresponds to the configuration where the ground truth is partially included in the graph. In the previous experiments, the value of k was fixed for every graph, but based on Figure 4.8, k could be adapted to the number of nodes. For example, if the number of nodes in the graph is six or lower, the candidate graph could be built with k set to 1 with the guarantee that the CG are inclusive (they contain their ground truth). Some value of k, specific to each number of nodes in the graphs lead to ICG with the smallest amount

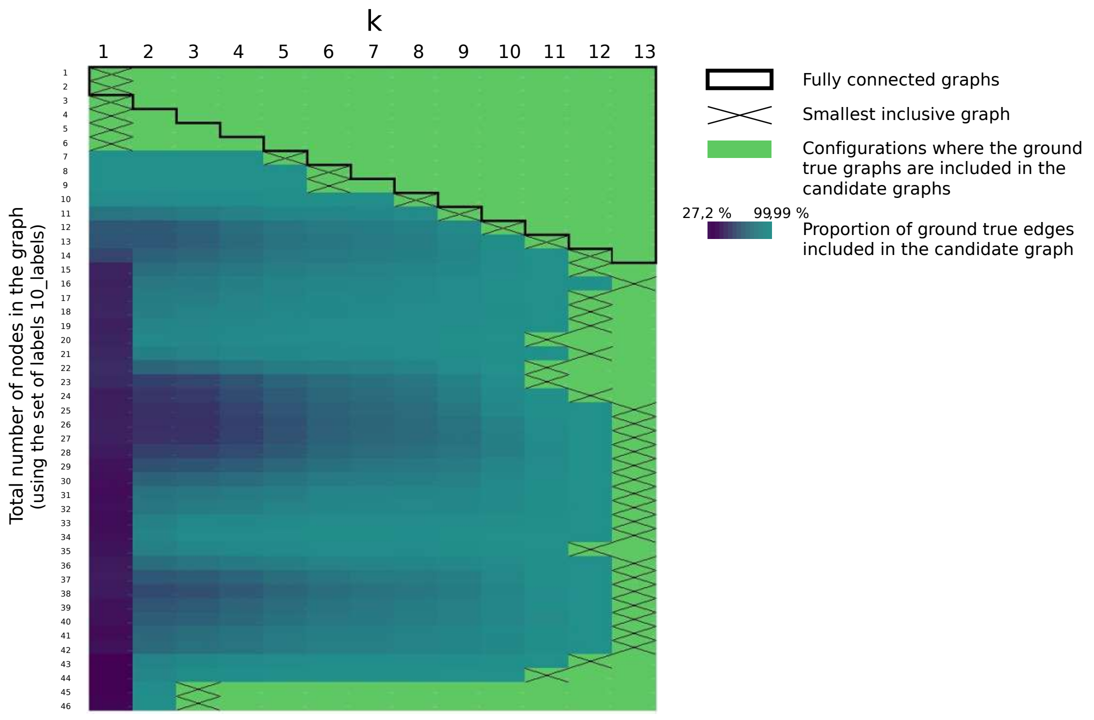


Figure 4.8: Average share of ground truth edges in the CG, by number of neighbours and number of primitives in the score using 10_labels class set on Musigraph

of negative edges. We call these graphs, *smallest inclusive graphs.* These graphs are indicated with a cross in Figure 4.8. For the graphs with one or two nodes, the candidate graphs are anyway fully connected, any values of k superior to one lead to a smallest inclusive graph. Having a larger value for k leads to the creation of avoidable negative edges. For example, when scores have 45 primitives, all the relations are between a node and its three closest neighbours. Building a candidate graph with k=13 introduces at least 450 negative edges compared to the smallest inclusive graph built with k=3 the minimal and inclusive k, which already ensures the candidate graph is inclusive. This technique offers an easy means to reduce the size of the CGs. However, it seems to be very specific to the Musigraph dataset and relies on an observation without any musical explanation. Additionally, some outcomes are unexpected and not easily explainable. For the graph with 45 and 46 nodes, the smallest inclusive graphs are obtained with k=3, which is very different from the one for 44 and fewer nodes. Nothing seems to explain this sudden change of behaviour. The small number of scores with this amount of primitive (see Figure 3.3) rather makes these graphs outliers. The actual expected profit of employing different values for k is also marginal. For scores with less than 15 primitives, the minimal and inclusive k is close to a value that would connect the KNN fully. For most scores with more than 15 primitives, the minimal and inclusive k is close to 13. For these reasons, this technique hasn't been implemented; a single value for k is used to generate all the candidate graphs.

Table 4.4 illustrates the performance of models learning on different values of k and being tested on graphs built for the same k and k=13 (ICG). These results have been obtained with the test procedure presented in Section 4.2.3. Note that it utilises the 10_labels set of classes and the CGs are pruned. For k=5, the MER increase a lot when testing on ICG but seems very decent on tests with k=5. On these graphs, around 15% of the ground truth edges are missing (see Figure 4.7), and the total number of edges is halved compared to CG when k=13 (see Figure 4.5). The MER is therefore increased by the ground truth edges missing in the CG and the error made on the CG edges (18% of prediction errors). These CGs have a small number of edges, therefore, the overall MER is low (0.28). The model trained on small graphs performs worse on larger graphs; the hypothesis that learning on smaller graphs could be relevant for predicting ICGs is rejected. On the contrary, having larger graphs at training seems to perform even better than testing and training with k=13. The best metrics are reached for training with k=14 and testing with k=13. Compared to the graph built with 13 neighbours, the graphs built with k=14 have, on average, 10% more edges. The low MER obtained for k=13 seems a bit surprising as this configuration corresponds to a dominant heuristic compared to the other configurations: Among the inclusive graphs, these graphs are the closest to the ground truth in terms of GED. Furthermore, the same experience repeated with a different learning rate scheduler gives 25% of MER; this instability is discussed in greater detail in Section 4.7. We can also state that the training time is not impacted by k.

Table 4.4: Influence of k on the semantic reconstruction and evaluation on ICG

k	Same k as training			ICG (k=13)			Training time
	Accuracy (%)	GED	MER	Accuracy (%)	GED	MER	
5	**82.05**	**5.69**	**0.28**	71.88	12.27	0.61	**4:14**
10	69.30	11.37	0.66	69.21	13.43	0.79	4:18
13	73.00	11.69	0.68	73.00	11.69	0.68	4:26
14	81.04	8.82	0.50	**81.61**	**8.48**	**0.49**	4:22
15	78.65	10.46	0.53	78.00	9.40	0.49	4:35
16	77.00	11.80	0.64	76.44	10.30	0.58	4:26

4.4.2 Semantic Pruning

Figure 4.6 shows that augmenting k leads to a higher proportion of negative edges. This proportion reaches 88% on average for CG based on non-pruned KNN graphs built with k=13 and the 10_labels class set. Some of these edges can be pruned without modifying the inclusiveness property of the candidate graphs

Leveraging the grammar of music notation, it appears that relationships between some primitive types are impossible. For example, two noteheads cannot be linked to one another in an errorless MuNG. Such links might be created in a KNN graph but can be

Table 4.5: Filter used to perform the semantic pruning when using the 10_labels class set. Any edge having a pair of endpoint labels different from those in the list is removed from the candidate graph.

Relations types
notehead - stem
notehead - beam
notehead - flag
notehead - legerLine
notehead - accidental
notehead - tie
notehead - others_slur_dynamics_etc
notehead - legerLine
notehead - tie
notehead - augmentationDot
rest - augmentationDot

pruned without modifying the inclusive property of a graph. To perform such pruning, a filter: a list of possible types of relation is associated with each class set. After creating a KNN graph, its edges are filtered and removed from the graph if they are not on the list. This pruning enables to get rid of a significant amount of negative edges. Figure 16 shows that for Musigraph, using the 10_labels class set and k=13, the proportion of negative edges drops by 17%. The relations types authorised by the filter associated with the 10_labels class set are listed in Table 4.5. To prune the graph further, it would be possible to establish a maximum distance between the nodes based on the relation type. For example, related beams and noteheads can be further distant than noteheads and stems, which are supposed to be in contact. These distances vary between the different fonts and handwritten notations, and because this work aims to be versatile, these methods have been rejected.

The filter used to prune the candidate graph depends on the granularity of the class set, and a more precise granularity can lead to better pruning. Some primitives grouped together in class sets with little precision disallow for accurate characterisation of certain relationships. In the 10_labels class set, all the noteheads are grouped under a single label; The associated filter must then allow the links between all the noteheads and the stems. But some noteheads, like whole noteheads, never make links with stems. The filter could prune these links if the granularity differentiates whole noteheads from the others. The filter is always defined using the minimal set of links that covers all the relations among the class set's primitives in the CWMN grammar.

Table 4.6 shows the result of the test procedure for pruned or unpruned graphs. The most notable observation is the relatively modest improvement in accuracy compared to the significant improvement of MER and GED. This suggests that while the models don't show an improvement in learning, outcomes are markedly improved.

Table 4.6: Influence of the semantic pruning on the semantic reconstruction

Metric / Pruning	Accuracy (%)	MER	GED
Prune	**84.19**	**0.47**	**7.35**
Not prune	82.07	2.47	38.61

4.4.3 Normalisation

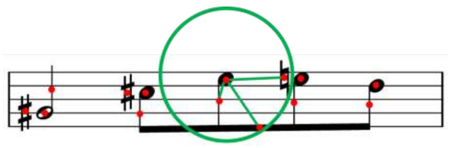
Normalising the data is a standard practice for any machine learning task. It ensures that each feature contributes equally to the learning process and usually boosts performance. This section discusses the normalisation of the MuNG node positions.

The primal idea is to map the positions and sizes of the primitives to a 0-1 range, but accounting for all the CWMN outlooks makes this task far from straightforward. Some scores might cover a full page, while a small measure is also a valid input for an OMR system. The size of the primitive could be used as a reference, but even the staff size varies from one typography to another. Our solution relies on dividing scores by measure and normalising them without using visual cues. We propose in section 3.2.3 an algorithm able to perform this division.

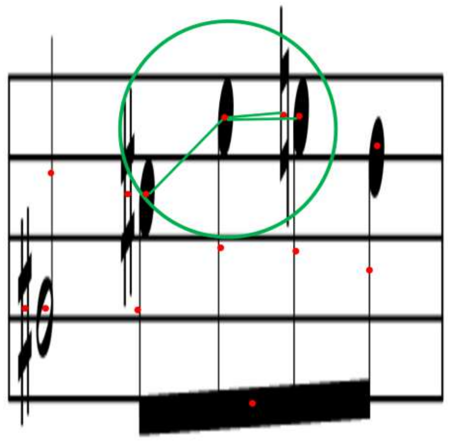
The process which divides the scores by measure also offers a negative edge filter. When constructing the KNN graph on a score with multiple staff and despite the semantic pruning, edges can be established between primitives from different staff. Such relations is impossible in an errorless MuNG.

From a measure and its image, the top-most, bottom-most, right-most and left-most bounding box edges are used to perform a min-max normalisation. Note that the four coordinates responsible for the size and position of primitives are normalised regardless of whether they represent a bounding box edge position, its centre, height or width. This seems natural for the bounding_box encoding configuration but less evident for the center_dimension encoding. To normalise the height and position, we would like to normalise them apart from the centre coordinates because they don't represent a coordinate. However, with the different fonts and the handwritten notation, primitive can have various sizes, which makes it impossible to normalise using a size of reference. Normalising height and width with the same parameter as the centre coordinates allows for a scaled system where, even normalised, the height can be summed to the x-centre parameter to obtain the lowest edge of the bounding box.

Normalising the size and positions modifies the feature matrix and isn't supposed to impact the adjacency matrix. Nevertheless, depending on the shape of the input, the KNN graphs built on a raw score might be different from the ones built on its normalised version. Normalising the input makes it a square: the top-most and bottom-most elements are as far apart as the right-most and the left-most. Because the edges are constructed using the Euclidean distance, on these '*squarified*' inputs, the closest neighbours of a node might be different from the ones in the original scores. Figure 4.7 illustrates this by



(a) Original score



(b) Normalised score

Figure 4.9: Representation of a normalised score and illustration of normalisation's impact on the KNN graph construction. The circles represent the distance between the node head and its three closest neigbhours, which differ in the two scores.

.

representing a normalised score and the three closest neighbours of a node head. The construction of the candidate graph has been shaped for non-distorted scores, to rely on the result obtained in the previous section, notably the value for k to get inclusive candidate graphs, normalisation must be performed after constructing the KNN graphs.

Table 4.7 shows the result of the test procedure for normalised or not scores. Depending on the configuration, normalisation can sometimes improve MER. Models perform best with normalisation when graphs are not pruned, although their overall performance remains poor. However, when applied to pruned graphs, normalisation worsens model performance. This outcome is understandable given the nature of the dataset, which consists of typeset scores where the distances between primitives or their sizes adhere to specific rules. For instance, the distance between a notehead and a potential accidental is usually consistent, but normalisation can cause these distances to vary significantly. As a result, the model loses the ability to learn effectively from these distance values. By default the scores are not normalised in the test procedure.

Table 4.7: Influence of the normalisation on semantic reconstruction

Prune	Normalisation \ Metric	Accuracy (%)	MER	GED
No	Not normalised	82.21	38.61	2.47
	Normalised	**87.34**	**27.41**	**1.97**
Yes	Not normalised	**81.26**	**9.07**	**0.54**
	Normalised	79.96	9.309	0.54

4.5 Model Architecture

Figure 4.10 illustrates the architecture of the model. The first module utilises GNNs to learn the graph nodes' representation. It takes as input a candidate graph and may use multiple layers from the three presented in the Background section (GeoGCN, GAT, or GraphSAGE). Only one type of layer is used in a GNN. The input dimensions of the first layer correspond to the width of the feature matrix, in our case, the number of labels in the class set used for the one hot encoding, plus four for the sizes and positions encoding. The output size is always 1024, and the hidden layers always have a size of 2048. The ReLU activation function is applied after each layer, and the final output is normalised. This module produces a node embedding matrix, which contains the representation of each graph node.

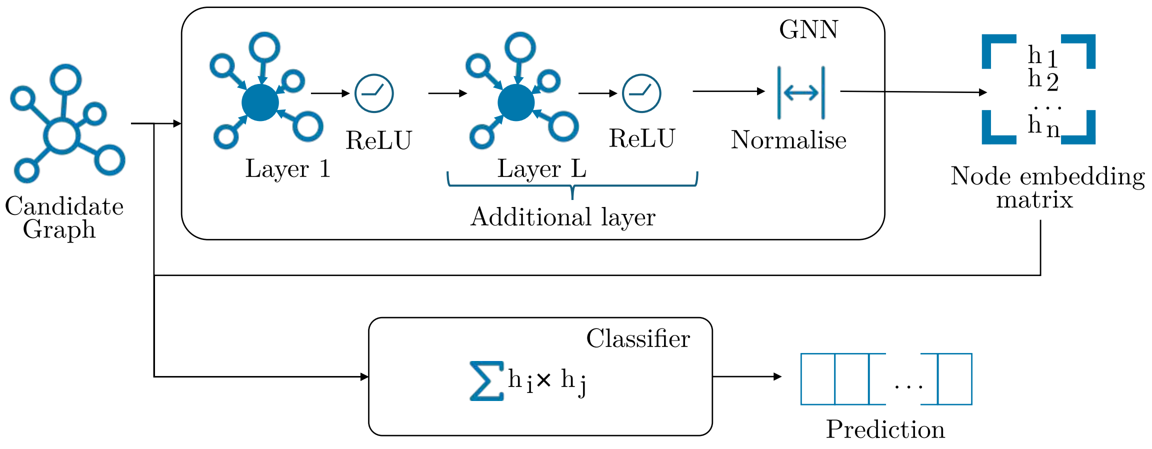


Figure 4.10: Model architecture

The second module is a simple classifier. It takes the node embedding matrix and the list of edges in the graph as input. For simplicity, the candidate graph edges are the only edges used at training. One could imagine using any set of edges, but this fashion is more natural. For each edge, the classifier sums the cross-product of the embeddings. Note that the obtained value is not modified further. At test time, values above 0.5 correspond to the prediction of an edge. This classifier remains the same for every model.

4.5.1 Using Directed MuNGs

Section 3.2.2 describes MuNGs in detail and states that they can be directed or not. Such a choice does not modify the musical information, but the behaviour of a GNN changes completely whether the graph's edges are considered directed or not.

When the edges are directed, by convention, GNNs consider the information to flow from the source to the target nodes. A node with an in-degree of 0 is updated only based on its own representation. For undirected edges, the information flows in both directions. In fact, in libraries like Pytorch Geometric, edges are always directed, but when the graphs are considered undirected, every edge is doubled with an edge going in the opposite direction. This has the consequence of making the number of edges less interpretable. For our problem, two edges form a single relationship between two primitives.

Some unexpected considerations should also be taken: To predict such undirected edges, it is natural to predict the existence of edges for both directions but with PyG, the two predictions might be different. The model might predict an edge to exist from a beam to a notehead but nothing prevents the opposite edges to be predicted the same. For our specific problem and because we haven't been further than constructing the MuNGs, we consider the edges separately. If an undirected edge is a positive edge, we consider its two directed edges to be true and treat them separately to compute the metrics. This problem should be solved with a system that would re-encode the music.

Table 4.8: Influence of the edges directed property on the semantic recon- struction

Metric / Edges	Accuracy (%)	GED	MER	Training time
Directed	82.21	38.61	2.47	**46:28**
Undirected	**90.57**	**11.30**	**0.70**	2:20:05

Table 4.8 shows the result of the test procedure for directed or undirected edges. Undirected edges greatly improve the metrics but at the cost of a much longer training time. By default, edges are undirected in the configuration of the test procedure.

4.5.2 Defining the Number of Layers

In GNNs, the number of layers defines how far the information will travel. With a single layer, a node's representation is updated only using the node's direct neighbours. With two layers, a node might perceive information from nodes that are two links away from it. Table 4.9 shows the result of the test obtained for different number of layers. For this test, only 25 epochs (unlike the usual 100) are used because the differences between the metrics are already significant: using three layers is by far the best compromise. This configuration results in the best accuracy, MER, and GER. Regarding the training times, using three layers seems to be acceptable compared to the fastest training and is much better than the configuration with one more layer.

Table 4.9: Influence of the number of layers on the semantic reconstruction

Number of layers	Accuracy (%)	MER	GED	Training time
1	75.85	0.68	11.25	**20:24**
2	76.64	0.61	9.95	26:88
3	**83.29**	**0.42**	**7.78**	35:43
4	80.11	0.49	9.27	97:08

4.5.3 Using Different Layers

Table 4.10 displays the results of the test procedure for the three kinds of layers presented in the background section. For this test, a few parameters are modified to obtain a more promising setup and compare the GNNs on a configuration closer to the final one. The CGs are pruned, their edges are considered undirected and the scheduler's patience is extended to 15 epochs. The GraphSAGE layers clearly outperform the two other kind of layers. With this configuration, the model using GAT layer didn't manage to learn properly and predicted every link to exist.

Table 4.10: Influence of the type of layer on the semantic reconstruction

Layer	**Accuracy (%)**	**MER**	**GED**
graphSAGE	**89.78**	**0.25**	**4.47**
geoGCN	81.79	0.36	7.95
GAT	30.80	2.07	30.26

4.6 Training Parameters

4.6.1 Loss Function

The loss function used to train the model is BCEWithLogitsLoss, which combines a Sigmoid layer and the Binary Cross Entropy (BCE) loss into a single class. This function expressed in Equation 4.12 is more *numerically stable* than using a plain Sigmoid function and a separate BCELoss. Using two functions one after the other propagates and amplifies the errors made because of the limited size of memory used to store the intermediate result. The combined implementation leverages the log-sum-exp trick, and treats the two functions as one, preventing potential issues that arise from floating-point precision errors [52]. Despite its technical advantages, BCEWithLogitsLoss is not inherently expressive. For the same model, a lower loss might result in worse predictions, making the comparison of loss values across different models delicate. Nevertheless, a significant advantage of BCEWithLogitsLoss is its positive weight parameter. It allows for the reweighting of positive examples in scenarios with imbalanced datasets. Usually, loss functions only allow for reweighting classes, with this parameter we can indicate that positive edges should be considered more important than negative ones.

$$L = \frac{1}{N}\sum_{i=1}^{N}\left[\log_e(1+e^{-z}) + z(1-y_i)\right] \tag{4.12}$$

Where N is the number of samples, y_i the true label and z_i the output of the model

4.6.2 Learning Rate Scheduler

The learning rate is managed using the ReduceLROnPlateau scheduler from the PyTorch library. An initial learning rate is specified in the configuration file. During training, if the validation loss does not improve for a specified number of iterations (defined by *scheduler_patience*), the learning rate is reduced by a factor (*scheduler_factor*), both of which are configurable parameters. By default, scheduler_patience is set to 10 and scheduler_factor to 0.5. This lessening of the learning rate allows us to start with a very high learning rate of 3 (by default), which helps the model learn faster.

When training models with various configurations, it is often observed that the loss initially decreases but then increases significantly. This pattern is seen in both the training and validation loss. To address this, we apply early stopping by selecting the best model with the lowest validation loss. Additionally, we introduced a mechanism called *jump_back_on_lr_change*. This feature ensures that whenever the scheduler adjusts the learning rate, the model weights revert to those from the best-performing configuration observed so far. This approach has proven effective in enhancing training stability and improving validation loss. Combining this mechanism with the scheduler is exactly similar to restarting a training from a checkpoint with a lower learning rate.

Figure 4.11 shows an example of learning curves with and without the option. By default, this mechanism is turned off.

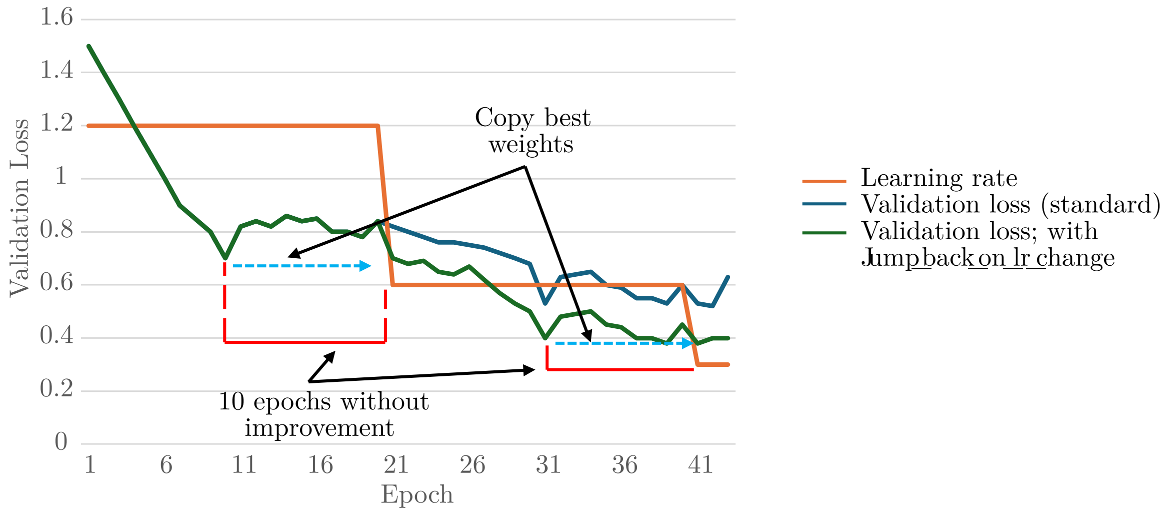


Figure 4.11: Learning rate evolution and associated mechanisms

4.7 Learning Instability and Reproducibility

In the course of our experiments, we encountered significant challenges related to the instability and reproducibility of results.

The first barrier to reproducible and stable results is inherent to PytorchGeometric. When executed on GPU, some operations are non-deterministic [53]. Although a seed is set to control many sources of randomness, quantifying the exact impact of the indeterministic operations remains challenging; On a GPU (NVIDIA 950M 2GB RAM), the same configuration can yield significantly different results, with accuracy varying by more than 50 points. However, this instability can be drastically reduced using a CPU, to say, it is even supposed to be deterministic. The tests conducted locally on CPU are deterministic but extremely slow. We used CPUs on a cluster offered by TU Wien where runs involving CPUs are close to being deterministic. Still, we could sometimes experience little differences of approximately 1% in accuracy for the same configuration.

The second barrier to reproducible and stable results is inherent to the GNNs. The seed we configure governs certain behaviours that are intended to be random or unaffected by the controlled sequence. For instance, the node aggregation order within a layer is dictated by the seed, yet this order is not expected to alter the layer's output. In a stable model, however, such randomness should not significantly influence the overall outcome. Table 4.11 presents the results of the test procedure for three different seeds (123, 1234, 12345) and the three different GNN layers. The default configuration has been altered to build undirected, pruned CG and the schedulers have a patience of 15 epochs and a factor of 0.7. According to P. Veličkovič, the instability should grow with the complexity of the layers [24]. GeoGCN is supposed to be more stable than GAT, which is supposed to be more stable than graphSAGE. In the table, the accuracy of the graphSAGE-bases models varies by more than 40%, and geoGCN by more than 10%. The only stable layer is the GAT, however, this configuration never managed to learn with the default configuration and always predicted every edge to exist.

Table 4.11: GNNs instability

Layer	Accuracy (%)	MER	GED
graphSAGE	48 - 90	0.25 - 1.26	4.47 - 22.59
geoGCN	78 - 89	0.23 - 0.44	4.96 - 7.95
GAT	31 - 31	2.07 - 2.07	30.26 - 30.26

4.8 Results

4.8.1 Defining a Promissing Setup

Table 4.12 presents the results obtained with the test procedure for a partial grid search combining four parameters. As the tests are realised on the same scores, the experiments are ordered using the GED, which gives more comparable values. This table summarises most of the tests presented earlier. For equivalent configuration, pruning is the only parameter that always leads to a better GED. The influence of the other parameters depends on the configuration of the pipeline: they might improve or make the GED worse. In this table, we also see the limit of the metric we use. The order obtained with the GED is different from the one we would obtain using the MER. This table shows GNNs can be very sensitive to their inputs. Changing one parameter in the pipeline that builds the graphs can lead to important differences in GED: Pruning the graph leads from the worst to the best GED. In this configuration, pruning only removes 17% of the negative edges. This grid search is very limited as many other parameters, like the class sets, the number of neighbours used to generate the KNN graphs, or the learning rate, remain fixed for every test. However, we can imagine promising configurations. Despite its visible drawbacks we arbitrarily keep the normalisation to obtain more versatile models.

Table 4.12: Partial grid search ordered by GED

Undirected edges	Normalised	Pruning	Size and position encoding*	GED	MER
			BB	38.61	2.47
			CD	30.56	2.12
	✓		CD	27.41	1.97
✓			BB	11.30	0.70
	✓	✓	BB	10.45	0.63
✓			CD	10.43	0.65
	✓	✓	CD	9.31	0.54
✓	✓	✓	BB	9.22	0.49
		✓	CD	9.07	0.53
✓	✓	✓	CD	8.819	0.50
✓		✓	BB	8.02	0.37
✓		✓	CD	7.48	0.35
		✓	BB	7.354	0.47

*BB: Bounding_box encoding. CD Centre_dimension encoding

4.8.2 Current Best Models Achieved for the 10_labels Class Set Across the Datasets with Single Models

It is important to acknowledge that achieving a 0% Graph Edit Distance or Music Error Rate is not a realistic expectation in this study. The datasets employed, such as Musigraph, inherently contain an unknown number of errors. For instance, in the Musigraph dataset, a specific relation is omitted in the score 52_syn_Beethoven; a connection between an empty notehead and a natural sign is missing. Additionally, errors may have been introduced during the dataset preparation process. In particular, the segmentation by measure of Muscima++ and DoReMi may have inadvertently introduced inaccuracies.

Despite the results obtained in Table 4.13, extended tests show better results when training on graphs constructed with k=13 for Musigraph. For the other dataset, we set k to 20. The algorithm used to divide the scores has a couple of drawbacks including that some one-page scores (notably for Muscima++) are considered as a single measure. In addition, the increased complexity of the scores makes 13 insufficient to obtain ICG. Setting k to 20 doesn't guarantee the CG to be inclusive either. In fact for Muscima_measure_cut, 80% of the ground true edges are included in the CG and for DoReMi_measure_cut the share of edges included in the CG is 91%. A better algorithm for dividing the scores by measure should be used to select a more meaningful value for k. The detailed pipeline configuration is available in Annexe B.

Table 4.13: Best models obtained for the different datasets with the 10_labels class set

Models	Dataset	k	Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	MER (%)	GED
model1	Musigraph	13	94.45	96.97	95.15	97.00	13.48	2.41
model2	DoReMi_measure_cut	20	89.71	71.20	84.75	91.00	37.02	8.39
model3	Muscima_measure_cut	20	84.37	64.71	72.00	88.00	38.08	13.87

4.8.3 Toward a Better Music Semantic Reconstruction

In this Section, we implement a solution based on multiple models and leverage a new set of classes.

The new class set is called 6_labels. It is available with its associated filter in Annex D; it offers a slightly more precise granularity than the 10_labels, e.g. it differentiates a notehead type from another. It also filters many primitives: dynamics and many rarer primitives are ignored. Nevertheless, it enables the encoding of most of the music and all the relationships from Musigraph.

The evaluation of the models includes the accuracy reached by the model for each link type. Table 4.14 presents the accuracy reached by a selection of models for the relation types corresponding to the 6_labels class set. Not all models perform equally across different link types; some exhibit superior performance for certain link types compared to others. Based on this observation, we can imagine an ensemble approach, where multiple models are employed together. During the testing phase, various models generate predictions.

For each link, depending on its specific type, we select the prediction from the model that has demonstrated the best performance for that particular link type.

Table 4.14: Accuracy obtained for each link type by a selection of models trained and evaluated on Musigraph for the 6_labels class set

Models	Layer	noteheadBlack - stem	noteheadBlack - accidental	noteheadBlack - Flag	noteheadBlack - Beam	Notehead WholeOrHalf - stem	noteheadWholeOrHalf - accidental
model4	graphSAGE	98.84	**99.00**	99.38	70.41	**98.80**	**90.87**
model5	graphSAGE	**98.89**	98.96	99.37	69.02	**98.80**	90.83
model6	geoGCN	91.52	90.53	90.34	**89.43**	96.63	89.99
model7	graphSAGE	98.85	98.97	**99.41**	68.45	98.61	90.83

Leveraging this strategy and combining the 4 models from the previous table, we obtained the results presented in Table 4.15. This strategy greatly improves the metrics obtained with a simple model in the previous section.

Table 4.15: Metrics obtained by the model ensemble

Dataset	k	Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	MER (%)	GED
Musigraph	13	97.09	97.76	92.70	99.06	6.09	0.70

CHAPTER 5

Conclusion

5.1 Training Insights

The main insight of the training process is the instability of the GNNs. This instability comes from different sources and has already been addressed in the literature. Klabunde et al. [54] demonstrated that GNNs based on different layers, but trained with the same data and presenting similar global metrics (same accuracy, Mean Absolute Error) could make their errors on very different nodes. This type of error motivates model ensemble solutions like the ones we built: Models employing GraphSAGE layers often struggle to predict relationships involving beams. In contrast, although models using Geo-GCN layers typically perform poorly overall, they manage to learn these specific types of relationships quite well. We experience that the instability is significant for any configuration and the seed heavily modifies the models' performances. On GPU even with similar configuration, significant differences can be experienced. Reproducibility and faith in a specific configuration are therefore altered and it's difficult to establish a "best setup". GNNs are very sensitive and might struggle to learn. For many configurations, despite a weight imbalance, the models ended up predicting every candidate edge to be connected or never connected.

GNNs sometimes learn best on unexpected configurations. We experienced that using three graphSAGE layers was best for the music semantic reconstruction. The main purpose of using multiple layers is to propagate information from a node through the graph. Considering the way music is usually read, this number of layers sounds odd. With inclusive candidate graphs, one layer could be sufficient to learn and predict the graph. It should not be needed to get information from so far away to predict an edge.

In this work, we also state that for a monophonic measure, a primitive is not related to another primitive beyond its 14th closest neighbour. This value depends on the definition of the MuNGs (what edges should be created). We didn't state any value for polyphonic scores as the process of dividing scores in measures sometimes creates very large scores that do not represent actual measures, and the value we could find would not be representative.

5.2 Discussion

The pipeline developed for this thesis involved several debatable decisions that have shaped both the approach's strengths and limitations.

The first criticism we can address is the limiting aspect of the solution regarding the set of classes. Our model relies on specific class sets and the primitives' labels must be one-hot encoded to form the feature vectors. Such a framework makes it impossible to adapt a pre-trained model for accepting new classes.

A critical area to address is the normalisation step. While intended to standardise musical scores for versatility, the approach was excessive. A more moderate strategy, using staff size as a reference for normalisation, would have aligned better with the inherent properties of the musical data and potentially improved the model's performance. An important majority of scores leverage typeset staff and considering them different from one to another is probably excessive.

Another point of criticism pertains to the parameter selection process for constructing the pipeline. The experiments were conducted with a fixed setup, and many conclusions drawn from this setup were applied to others. For instance, while three layers of graphSAGE appeared optimal in this particular setup, it was assumed that this number of layers would also be ideal for other types of layers. However, the assumption that these parameters would generalise well across different configurations is likely overly optimistic. The observed sensitivity of the model suggests that allowing more flexibility in parameter tuning could potentially uncover more effective configurations.

Despite these criticisms, the testing framework itself is robust and provides a solid foundation for evaluating model performance. It is bounded by the metrics extensively discussed in Section 4.2. This allows for an objective comprehensible measurement of how good the models are. In summary, the music semantic reconstruction method developed here shows limited performance, but a better combination of parameters could probably improve it.

5.3 Answer to the Research Questions

1. How suitable (5% of MER or lower) are GNNs to solve the music semantic reconstruction stage?

   The best MER we achieved is on Musigraph and is just above 6%. If the baseline set by Baro et al. [19] hasn't been reached, no upper bound for GNNs has been found during this work. The comparison to the model obtained by Baro et al is not trivial. First, because the 5% they obtained includes the error made by their music object detector. We used the ground true object as input and our MER only reflects the error made during the reconstruction stage. Then because the model they use is very specific to the font used in Musigraph: They include knowledge based on the distance between the objects to prune their candidate graphs. We also want to mention that their model architecture (mostly kept private) might be very different from ours. They obtained their final weights after 5 iterations while our model requires a few hundred.

2. How does such a model handle different input configurations, particularly in terms of musical complexity (amount of musical information on a score like multiple instruments, accompanied melody...)?

   Without a perfect metric, it is not possible to properly answer this question. We can state that our models output better MER on Musigraph which only contains monophonic scores than on DoReMi and Muscima++. But as discussed in Section 4.2, comparing MER on different scores remains delicate. Nevertheless, the number of errors made on the more complex scores is significantly more important than on the score from Musigraph.

3. How well do GNN-based approaches for semantic reconstruction scale with an increasing number of input primitives?

   As discussed in Section 4.4.3, this question doesn't make a lot of sense. Scores can be split and reassembled. The few models trained on non-split scores showed very poor results but only a few configurations have been tried.

4. How does the GNN initialisation (fully connected, based on explicit rules) impact the link prediction?

   The best models we obtained are based on the smallest ICG. The tests specifically to observe the impact of pruning the candidate graphs show significant differences and that having a limited number of links helps the model to learn. This statement is to be considered carefully as Table 4.4 shows that the lightest graphs don't always lead to the best result. The instability of the model makes it hard to answer this question but we can state that in most situations smaller graphs lead to better results.

5.4 Future Work

This work could be further extended in several important areas. One key aspect is the development of a new metric specifically for evaluating the complexity of musical notation. Creating these metrics would provide a more objective way to assess the performance of OMR systems and allow for a fair comparison between models performing on different datasets for example.

To aim for a better model, several promising directions were identified during this research. For instance, exploring alternative distance metrics, such as the distance between the primitives' bounding box rather than their centre, could yield improvements in model performance. Additionally, implementing automated model ensemble learning systems could enhance overall performance by integrating multiple models and leveraging their collective strengths.

This pipeline could also be altered to leverage novel GNN-inspired models. The current research has highlighted the potential for improving link prediction models by transforming the task into a graph classification problem. To perform such transformation the SEAL framework has become popular [55]. This framework is used for link prediction based on the enclosing subgraph of the potential link endpoints. We experienced the best performances for models using 3 layers, which allows the information of a node to travel across three links to build the final node embeddings. In theory, it could be possible to predict a link solely based on the information contained by the direct neighbours of the link endpoints. This number of layers might indicate that giving an overview of the link environment helps the model. As SEAL integrates this enclosing graph directly, it might be specifically interesting for music semantic reconstruction. Further advanced variations of this framework transform the task further and convert the graph classification problems into node classification tasks [56]. These model architectures have demonstrated state-of-the-art performance in other domains and could potentially be applied to music semantic reconstruction. To keep more common GNN architectures, the heterogeneous graphs neural networks could also be interesting candidates. These GNNs can treat nodes differently depending on their class [57].

Despite the goal of achieving versatile models, none have been trained on multiple datasets. This is primarily due to the significant differences in how MuNGs are defined across datasets, which would require substantial reannotation to establish a consistent ground truth (specifically to align DoReMi to the other datasets). Additionally, the stark imbalance in dataset sizes raises concerns about the effectiveness of such a merge. For example, incorporating the 140 scores from Muscima++ into the 18,921 scores from Musigraph would likely have a negligible impact on model performance. Future work could focus on conducting a deeper investigation into how structural complexity influences the performance of GNN-based models for music notation semantic reconstruction.

Overview of Generative AI Tools Used

In this thesis, generative AI tools such as Grammarly and ChatGPT (versions 3 and 4) were employed to refine and reformulate specific sentences, ensuring clarity and coherence in the writing. Additionally, DeepL was used for translation tasks and to identify suitable synonyms, enhancing the precision and fluidity of the language. These tools served as assistants, aiding in the refinement of the text, but were never used to generate large blocks of content.

List of Figures

List of Tables

Acronyms

BCE	Binary Cross Entropy. 52
CG	Candidate Graph. 25
CWMN	Common Western Music Notation. 1
GED	Graph Edit Distance. 30
Geo-GCN	Geometric Graph Convolutional Neural Network. 12
GNN	Graph Neural Network. 5
ICG	Inclusive Candidate Graph. 27
KNN	K-Nearest Neighbors. 26
MEI	Music Encoding Initiative. 2
MER	Music Error Rate. 6
MIDI	Musical Instrument Digital Interface. 2
MLP	Multi-Layer perceptron. 13
MP	Message Passing. 11
MS	Mesasure's Surface. 22
MuNG	Music Notation Graph. 5
OMR	Optical Music Recognition. 4
PMS	Primitives on the Measure's Surface. 22
POMS	Primitives Out of the Measure's Surface. 22
PyG	PyTorch Geometric. 14

Bibliography

[1] C. Delaplace. The etymologies of isidore of seville, translated with introduction and notes by s.a. barney, w.j. lewis, j.a. beach, o. berghof, with the collaboration of m. hall. *Anabases [Online]*, 14, 2011. accessed on 26 August 2024.

[2] M.L. West. The babylonian musical notation and the hurrian melodic texts. *Music & Letters*, 75(2):161–179, 1994.

[3] A. Barbera. Octave species. *The Journal of Musicology*, 3(3):229–241, 1984.

[4] J Raasted. The hagiopolites: A byzantine treatise on musical theory, preliminary edition. *Cahiers de l'Institut du Moyen-Âge Grec Et Latin*, 45:1–99, 1983.

[5] O. Wright. *The Modal System of Arabian and Persian Music, 1250-1300: An Interpretation of Contemporary Texts*. SOAS University of London, 1969.

[6] I.D. Bent. Musical notation, 2024. Encyclopedia Britannica.

[7] A.H. King. *Four Hundred Years of Music Printing*. British Museum, 1968.

[8] J. Maxwell and S.M. Ornstein. Mockingbird: A composer's amanuensis. In *Proceedings of the 1981 International Computer Music Conference, ICMC 1981, Denton, Texas, USA, 1981*. Michigan Publishing, 1981.

[9] MakeMusic, Inc. Finale, 1988. [Computer Software]. https://www.finalemusic.com.

[10] Muse Group. Musescore, 2002. [Computer Software]. https://www.musescore.org.

[11] D. Kastrup, W. Lemberg, H. Nienhuys, J. Nieuwenhuizen, C. Sorensen, and J. et al. Warchoł. Lilypond, 1997. [Computer Software]. https://lilypond.org/.

[12] Muse Group. Staffpad, 2015. [Computer Software]. https://www.staffpad.net/.

[13] Music Encoding Initiative. Music Encoding Initiative (MEI), 2010. https://music-encoding.org.

[14] The MIDI Association. Musical Instrument Digital Interface (MIDI), 1983. https://www.midi.org.

[15] M. Good. MusicXML, 2000. https://www.musicxml.com.

[16] J. Calvo-Zaragoza, J. Hajič jr., and A. Pacha. Understanding optical music recognition. *ACM Comput. Surv.*, 53(4), jul 2020.

[17] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A R. S. Marçal, C. Guedes, and J. S. Cardoso. Optical music recognition: state-of-the-art and open issues. *Int. J. Multim. Inf. Retr.*, 1(3):173–190, 2012.

[18] A. Pacha, J. Calvo-Zaragoza, and J. Hajič jr. Learning notation graph construction for full-pipeline optical music recognition. In *Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR 2019)*, pages 75–82, 2019.

[19] A. Baró, P. Riba, and A. Fornés. Musigraph: Optical music recognition through object detection and graph neural network. In Utkarsh Porwal, Alicia Fornés, and Faisal Shafait, editors, *Frontiers in Handwriting Recognition*, pages 171–184, Cham, 2022. Springer International Publishing.

[20] J. Hajič jr., M. Kolárová, A. Pacha, and J. Calvo-Zaragoza. How current optical music recognition systems are becoming useful for digital libraries. In *Proceedings of the 5th International Conference on Digital Libraries for Musicology*, DLfM '18, page 57–61, New York, NY, USA, 2018. Association for Computing Machinery.

[21] M. Schobrun. *The Everything Reading Music Book: A Step-By-Step Introduction to Understanding Music Notation And Theory*. Everything series. Adams Media, Avon, MA, 2005.

[22] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[23] G. Chartrand. *Introduction To Graph Theory*. Walter Rudin student series in advanced mathematics. McGraw-Hill Education (India) Pvt Limited, 2006.

[24] P Veličković. Everything is connected: Graph neural networks. *CoRR*, abs/2301.08210, 2023.

[25] M.M. Bronstein, J. Bruna, T. Cohen, and P Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021.

[26] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[27] Y. Zhang, P. Tiño, A. Leonardis, and K. Tang. A survey on neural network interpretability. *CoRR*, abs/2012.14261, 2020.

[28] A. McCallum. Cora, 2024. [Data set]. https://dx.doi.org/10.21227/jsg4-wp31.

[29] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R.P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[30] Y. Feng and S. Zhang. Prediction of drug-drug interaction using an attention-based graph neural network on drug molecular graphs. *Molecules*, 27(9), 2022.

[31] P. Spurek, T. Danel, J. Tabor, M. Smieja, L. Struski, A Slowik, and L. Maziarka. Geometric graph convolutional neural networks. *CoRR*, abs/1909.05310, 2019.

[32] P Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L Kaiser, and I Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[34] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[35] M. Fey and J.E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[36] J. Hajič jr. and P. Pecina. The muscima++ dataset for handwritten optical music recognition. In *14th International Conference on Document Analysis and Recognition, ICDAR 2017*, pages 39–46, Kyoto, Japan, November 13-15 2017.

[37] A. Fornés, A. Dutta, A. Gordo, and J. Lladós. Cvc-muscima: a ground truth of handwritten music score images for writer identification and staff removal. *Int. J. Document Anal. Recognit.*, 15(3):243–251, 2012.

[38] E. Shatri. DoReMi, 2021. [Data set]. https://elonashatri.github.io/doremi-dataset.html.

[39] Steinberg Media Technologies. SMuFL, 2024. https://www.smufl.org/.

[40] D. Spreadbury and R. Piéchaud. Standard music font layout (smufl). In *First International Conference on Technologies for Music Notation and Representation - TENOR2015*, pages 146–153. Institut de Recherche en Musicologie, 2015.

[41] Steinberg. Dorico, 2016. [Data set]. https://www.steinberg.net/dorico/.

[42] J. Hajič jr., M. Dorfer, G. Widmer, and P. Pecina. Towards full-pipeline handwritten omr with musical symbol detection by u-nets. In *International Society for Music Information Retrieval Conference*, 2018.

[43] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R.R. Salakhutdinov, and A.J. Smola. Deep sets. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[44] A. Kazi, L. Cosmo, S. Ahmadi, N. Navab, and M.M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1606–1617, February 2023.

[45] H. Sáez de Ocáriz Borde, A. Kazi, F. Barbero, and P. Liò. Latent graph inference using product manifolds, 2023.

[46] J. Hajič jr. A case for intrinsic evaluation of optical music recognition. *1st International Workshop on Reading Music Systems*, page 15–16, 2018.

[47] T. Raja de Reuse. *Detecting Errors in Optical Music Recognition Output with Machine Learning.* Phd dissertation, McGill University, April 2024.

[48] P. Wills and F.G. Meyer. Metrics for graph comparison: A practitioner's guide. *PLOS ONE*, 15(2):1–54, 02 2020.

[49] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, 2010.

[50] E. Holder, E. Tilevich, and A. Gillick. Musiplectics: computational assessment of the complexity of music scores. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, Onward! 2015, page 107–120, New York, NY, USA, 10 2015. Association for Computing Machinery.

[51] S. Chiu and M. Chen. A study on difficulty level recognition of piano sheet music. In *2012 IEEE International Symposium on Multimedia*, pages 17–23, 2012.

[52] PyTorch Contributors. *torch.nn.BCEWithLogitsLoss*, 2024. Accessed: 2024-07-24.

[53] PyTorch Contributors. *Randomness in PyTorch*, 2024. https://pytorch.org/docs/stable/notes/randomness.html.

[54] M. Klabunde and F. Lemmerich. On the prediction instability of graph neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 187–202. Springer Nature Switzerland, 2023.

[55] L. Cai and S. Ji. A multi-scale approach for graph link prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3308–3315, Apr. 2020.

[56] L. Cai, J. Li, J. Wang, and S. Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:5103–5113, 2022.

[57] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan. Simple and efficient heterogeneous graph neural network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):10816–10824, Jun. 2023.

APPENDIX A

Datasets

This first appendix gives more details on the datasets presented in Chapter 3.

Table A.1: Muscima++ labels and their number of appearances in the dataset

Labels	Counts	Labels	Counts
noteheadBlack	135583	articTenutoAbove	332
stem	120339	articAccentAbove	316
kStaffLine	33960	accidentalDoubleSharp	308
beam	28788	timeSig3	298
rest16th	28284	dynamicForte	298
flag16thUp	17931	timeSig2	293
barline	16467	articTenutoBelow	284
rest8th	14247	accidentalDoubleFlat	240
rest32nd	10001	restHalf	233
flag32ndUp	7244	articMarcatoAbove	191
gClef	6173	cClef	156
accidentalFlat	6056	dynamicPP	141
accidentalSharp	6052	dynamicSforzato	132
accidentalNatural	4592	timeSignatureComponent	115
slur	3586	flag16thDown	113
tie	3454	dynamicMF	96
augmentationDot	2463	accidentalQuarterToneSharpStein	86
restQuarter	2191	timeSig5	77
noteheadHalf	2138	accidentalQuarterToneFlatStein	73
flag8thUp	1817	dynamicFF	63
articStaccatoAbove	1339	dynamicText	50
tupletText	1015	dynamicMP	45
flag8thDown	1010	timeSig6	40
fClef	927	timeSig7	37
gradualDynamic	875	ornamentTrill	34
articStaccatoBelow	867	timeSigCommon	34
systemicBarline	859	dynamicFortePiano	28
tupletBracket	752	accidentalThreeQuarter - TonesSharpStein	26
timeSig4	573	timeSig9	25
timeSig8	518	articMarcatoBelow	21
noteheadWhole	493	dynamicForzando	18
articStaccatissimoAbove	452	dynamicFFF	18
articAccentBelow	369	dynamicPPP	14
dynamicPiano	351	flag32ndDown	12
restWhole	336	timeSigCutCommon	8
articStaccatissimoBelow	332		

Table A.2: Muscima++'s 80 most present relation types and number of appearances in the dataset

Relation types	Counts	Relation types	Counts
noteheadFull - stem	21451	noteheadFullSmall - stem	351
noteheadFull - staff	21333	noteheadFullSmall - staff	348
noteheadFull - beam	18083	dynamicsText - dynamicLetterP	330
noteheadFull - staffSpace	8743	noteheadFullSmall - beam	330
noteheadFull - staffLine	8201	clefF - staff	284
noteheadFull - slur	7889	noteheadFull - accidentalFlat	284
noteheadFull - legerLine	6348	otherText - characterSmallE	274
staff - staffSpace	5298	repeat - repeatDot	266
measureSeparator - staff	4549	noteheadFull - flag16thDown	235
staff - staffLine	4415	noteheadFull - flag16thUp	227
measureSeparator - barline	3204	tuple - numeral3	225
noteheadFull - articulationStaccato	1628	restHalf - staff	216
noteheadHalf - stem	1497	otherText - characterSmallC	212
noteheadHalf - staff	1497	stem - multipleNoteTremolo	212
noteheadFull - augmentationDot	1449	noteheadFull - noteheadFullSmall	209
noteheadFull - accidentalSharp	1355	clefC - staff	194
noteheadFull - flag8thDown	1204	otherText - characterSmallO	193
rest8th - staff	1134	timeSignature - staff	192
noteheadFull - tie	967	noteheadFull - articulationAccent	190
noteheadFull - flag8thUp	957	otherText - characterSmallS	176
noteheadFull - accidentalNatural	948	ornamentTrill - characterSmallR	174
noteheadFull - tuple	805	noteheadHalf - noteheadFullSmall	173
restQuarter - staff	803	ornamentTrill - characterSmallT	171
noteheadFull - dynamicCrescendoHairpin	747	noteheadWhole - staff	171
keySignature - accidentalFlat	731	tempoText - characterSmallE	169
keySignature - staff	695	noteheadFullSmall - flag8thUp	169
noteheadHalf - staffSpace	624	noteheadFullSmall - legerLine	162
keySignature - accidentalSharp	614	otherText - characterDot	162
staffGrouping - staff	611	noteheadFullSmall - slur	156
noteheadHalf - staffLine	566	restWhole - staff	153
noteheadFull - dynamicDiminuendoHairpin	541	noteheadFull - articulationTenuto	150
noteheadHalf - augmentationDot	505	otherText - characterSmallR	140
dynamicsText - characterSmallF	494	timeSignature - numeral4	139
noteheadHalf - legerLine	439	tempoText - characterSmallO	133
rest16th - staff	436	noteheadFullSmall - staffSpace	133
dynamicsText - dynamicLetterF	405	repeat - barline	124
clefG - staff	402	staffGrouping - staffGrouping	121
dynamicsText - characterSmallP	395	noteheadHalf - ornamentTrill	116
noteheadHalf - tie	393	staffGrouping - brace	115
noteheadHalf - slur	366	rest8th - augmentationDot	112

Table A.3: Musigraph labels and their number of appearances in the dataset

Labels	Counts
stem	98417
notehead-full	91338
thin_barline	37505
flat	24272
sharp	23962
16th_flag	20084
beam	16068
8th_flag	12560
natural	10957
notehead-empty	7309
quarter_rest	4790
half_rest	4484
16th_rest	4436
8th_rest	3358
c-clef	2970
timeSig_2-2	2837
timeSig_cut	2595
f-clef	2462

Table A.4: Musigraph relation types and number of appearances in the dataset

Relation types	Counts
notehead-full - stem	91338
notehead-full - beam	64218
notehead-full - flat	22472
notehead-full - sharp	22155
notehead-full - 16th_flag	20110
notehead-full - 8th_flag	12560
notehead-full - natural	10083
notehead-empty - stem	7079
notehead-empty - sharp	1807
notehead-empty - flat	1800
notehead-empty - natural	1

Table A.5: DoReMi labels and their number of appearances in the datase

Labels	Counts	Labels	Counts
noteheadBlack	135583	articTenutoAbove	332
stem	120339	articAccentAbove	316
kStaffLine	33960	accidentalDoubleSharp	308
beam	28788	timeSig3	298
rest16th	28284	dynamicForte	298
flag16thUp	17931	timeSig2	293
barline	16467	articTenutoBelow	284
rest8th	14247	accidentalDoubleFlat	240
rest32nd	10001	restHalf	233
flag32ndUp	7244	articMarcatoAbove	191
gClef	6173	cClef	156
accidentalFlat	6056	dynamicPP	141
accidentalSharp	6052	dynamicSforzato	132
accidentalNatural	4592	timeSignatureComponent	115
slur	3586	flag16thDown	113
tie	3454	dynamicMF	96
augmentationDot	2463	accidentalQuarterToneSharpStein	86
restQuarter	2191	timeSig5	77
noteheadHalf	2138	accidentalQuarterToneFlatStein	73
flag8thUp	1817	dynamicFF	63
articStaccatoAbove	1339	dynamicText	50
tupletText	1015	dynamicMP	45
flag8thDown	1010	timeSig6	40
fClef	927	timeSig7	37
gradualDynamic	875	ornamentTrill	34
articStaccatoBelow	867	timeSigCommon	34
systemicBarline	859	dynamicFortePiano	28
tupletBracket	752	accidentalThreeQuarterTonesSharpStein	26
timeSig4	573	timeSig9	25
timeSig8	518	articMarcatoBelow	21
noteheadWhole	493	dynamicForzando	18
articStaccatissimoAbove	452	dynamicFFF	18
articAccentBelow	369	dynamicPPP	14
dynamicPiano	351	flag32ndDown	12
restWhole	336	timeSigCutCommon	8
articStaccatissimoBelow	332		

Table A.6: DoReMi relation types and number of appearances in the dataset

Relation types	Count
noteheadBlack - stem	135850
noteheadBlack - beam	96265
noteheadBlack - slur	17488
noteheadBlack - tie	5437
noteheadBlack - tupletText	4762
noteheadBlack - tupletBracket	2548
noteheadHalf - stem	2379
noteheadBlack - articStaccatoAbove	2341
noteheadBlack - articStaccatoBelow	1484
noteheadBlack - articStaccatissimoAbove	1044
noteheadBlack - articStaccatissimoBelow	700
noteheadBlack - articAccentAbove	660
noteheadHalf - slur	629
noteheadBlack - articAccentBelow	594
noteheadHalf - tie	543
noteheadBlack - articTenutoAbove	520
noteheadBlack - articTenutoBelow	492
noteheadBlack - articMarcatoAbove	401
noteheadWhole - tie	161
noteheadHalf - articAccentBelow	78
noteheadHalf - articTenutoAbove	62
noteheadWhole - slur	60
noteheadHalf - articMarcatoAbove	56
noteheadHalf - articTenutoBelow	27
noteheadHalf - articAccentAbove	26
noteheadBlack - articMarcatoBelow	23
noteheadWhole - articTenutoBelow	14
noteheadWhole - articTenutoAbove	9
noteheadHalf - articMarcatoBelow	7
noteheadHalf - articStaccatoBelow	2
noteheadWhole - articAccentBelow	1

APPENDIX B

Test Configuration

This second appendix contains the pipeline configurations used during the tests and final training of the models. These configurations are consistent with those employed in the implementation, though the terminology may slightly differ from that used in the master thesis. We begin with a brief explanation of each configuration parameter.

- **dataset**: Name of the dataset to use.
- **labels_to_use**: Set of classes used to one-hot-encode the labels.
- **position_as_bounding_box**: If True, the bounding_box encoding is used, if False, the centre_dimension.
- **n_neighbors_KNN**: Number of neighbours used to generate the KNN Graph (k).
- **prefilter_KNN**: If True, semantic pruning is performed.
- **normalize_positions**: If True, normalise the score.
- **seed**: Value for the random seed.
- **learning_rate**: Starting value of the learning rate.
- **load_model**: Path to the model to load and use as checkpoint.
- **start_from_best_model**: If True and a path is provided in *load_model* the learning uses the best model of the checkpoint, if False it uses the last model (continue the learning). If the parameter load_model is False, this parameter is ignored. In the following configurations, the paths are replaced by *path* if they were set.
- **scheduler_factor**: Factor for the learning rate scheduler.
- **scheduler_patience**: Patience for the learning rate scheduler.

- **undirected_edges**: If True, consider the edges to be undirected.
- **jump_back_on_lr_change**: If True, the model weights are modified to correspond to the best model obtained so far when the lr is updated.
- **layer_type**: Type of layer ('graphSAGE', 'GAT' or 'geoGCN'.
- **batch_size**: Batch size.
- **weight_imbalance**: Weight given to the loss function, if its value is -1, the weight is computed on the training test.
- **n_epochs**: Number of epochs.
- **visualize_first_score**: Generate a visual representation of the graph predicted for a test score.
- **save_model**: If True, save the model as a checkpoint.
- **model_config**: Description of the model layer by layer (employ the ___str___ function of the layers).
- **fine_tunning**: [OPTIONNAL] if True, the best model is updated anyway during the training: the best model is the best model obtained during the training even if its validation loss value is superior to the checkpoint one.
- **starting_epoch**: [OPTIONNAL] This configuration parameter is generated automatically. If the model starts from a checkpoint, the starting epoch is saved here.

Test Procedure configuration

The baseline for the test procedure is defined by the following parameters. For each test one or a few parameters are modified.

- **dataset**: 'musigraph'
- **labels_to_use**: '10_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 14
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 123
- **learning_rate**: 3
- **load_model**: False
- **start_from_best_model**: True
- **scheduler_factor**: 0.5
- **scheduler_patience**: 10
- **undirected_edges**: False
- **jump_back_on_lr_change**: False
- **layer_type**: 'graphSAGE'
- **batch_size**: 1024
- **weight_imbalance**: -1
- **n_epochs**: 100
- **visualize_first_score**: True
- **save_model**: True
- **model_config**: " 'in_channels': 14, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"

The following models correspond to the configuration of the seven models selected for the final results. These models have been obtained after reusing a few checkpoints and pusuing the training from them. The checkpoints have always the same configuration as the models that pursued their training. Note that these models are trained on GPU and their training is not perfectly reproducible.

model1

- **dataset**: 'musigraph'
- **labels_to_use**: '10_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 13
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 12345
- **learning_rate**: 3
- **load_model**: False
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 15
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 1024
- **weight_imbalance**: 3.2722200546234883
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **model_config**: " 'in_channels': 14, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"

model2

- **dataset**: 'doremi_measure_cut'
- **labels_to_use**: '10_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 20
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 12345
- **learning_rate**: 3
- **load_model**: './models/doremi_08-12-2024_11-40.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 2048
- **weight_imbalance**: 5.577372875211054
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False

- **starting_epoch**: 1814
- **model_config**: " 'in_channels': 14, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"

model3

- **dataset**: 'doremi_measure_cut'
- **labels_to_use**: '10_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 20
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 12345
- **learning_rate**: 3
- **load_model**: './models/doremi_08-12-2024_11-40.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 2048
- **weight_imbalance**: 5.577372875211054
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False
- **starting_epoch**: 1814
- **model_config**: " 'in_channels': 14, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"

model4

- **dataset**: 'musigraph'
- **labels_to_use**: '6_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 13
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 123
- **learning_rate**: 3
- **load_model**: './models/musigraph_6lab_08-12-2024_11-52.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 2048
- **weight_imbalance**: 3.278202627129666
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False

- **model_config**: " 'in_channels': 10, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"
- **starting_epoch**: 596

model5

- **dataset**: 'musigraph'
- **labels_to_use**: '6_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 13
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 123
- **learning_rate**: 3
- **load_model**: './models/musigraph_6lab_08-12-2024_12-26.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 2048
- **weight_imbalance**: 3.278202627129666
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False
- **model_config**: " 'in_channels': 10, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"
- **starting_epoch**: 808

model6

- **dataset**: 'musigraph'
- **labels_to_use**: '6_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 13
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 123
- **learning_rate**: 3
- **load_model**: './models/musigraph_6lab_08-12-2024_13-01.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'graphSAGE'
- **batch_size**: 2048
- **weight_imbalance**: 3.278202627129666
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False

- **model_config**: " 'in_channels': 10, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 2048, 'aggr': 'mean', 'in_channels': 2048, 'out_channels': 1024, 'aggr': 'mean'"
- **starting_epoch**: 882

model7

- **dataset**: 'musigraph'
- **labels_to_use**: '6_labels'
- **position_as_bounding_box**: True
- **n_neighbors_KNN**: 13
- **prefilter_KNN**: True
- **normalize_positions**: True
- **seed**: 12345
- **learning_rate**: 1
- **load_model**: './models/musigraph_geo_08-10-2024_12-51.pth'
- **start_from_best_model**: True
- **scheduler_factor**: 0.7
- **scheduler_patience**: 30
- **undirected_edges**: True
- **jump_back_on_lr_change**: True
- **layer_type**: 'geoGCN'
- **batch_size**: 2048
- **weight_imbalance**: 3.278202627129666
- **n_epochs**: 250
- **visualize_first_score**: True
- **save_model**: True
- **fine_tunning**: False
- **starting_epoch**: 700
- **model_config**: " 'coors': 2, 'in_channels': 10, 'out_channels': 2048, 'hidden_size': 1, 'dropout': 0, 'coors': 2, 'in_channels': 2048, 'out_channels': 2048, 'hidden_size': 1, 'dropout': 0, 'coors': 2, 'in_channels': 2048, 'out_channels': 1024, 'hidden_size': 1, 'dropout': 0"

APPENDIX C

KNN configuration

n_value	10_labels / pruned			10_labels / not pruned			all_musigraph_labels / not pruned		
	%positive_ edges	%positive_ edges_covered	count_edges	%positive_ edges	%positive_ edges_covered	count_edges	%positive_ edges	%positive_ edges_covered	count_edges
1	99.293	48.490	123857	62.947	48.490	195372	50.333	46.441	234012
3	78.746	75.903	244467	33.584	75.903	573206	29.229	75.902	658615
5	57.636	84.083	370000	22.618	84.082	942827	19.481	84.084	1094677
7	46.577	88.872	483935	17.801	88.872	1266181	15.030	88.872	1499695
9	38.491	92.554	609848	14.883	92.554	1577209	12.486	92.531	1879540
11	34.551	98.746	724840	13.351	98.746	1875773	11.195	98.699	2236122
13	30.456	100.000	832757	11.789	100.000	2151312	9.873	99.999	2568922
14	28.631	100.000	885832	11.119	100.000	2281023	9.304	100.000	2725896

APPENDIX D

6_labels class set

Table D.1: 6_labels granularity mapping

Musigraph	Muscima++	DoReMi	6_labels
stem	stem	stem	stem
notehead-full	noteheadFull	noteheadBlack	noteheadBlack
notehead-empty	noteheadHalf, noteheadFullSmall, noteheadWhole	noteheadHalf, noteheadWhole	noteheadWholeOrHalf
beam	beam	beam	beam
sharp, flat, natural	augmentationDot, accidentalSharp, accidentalFlat, accidentalNatural, accidentalDouble Sharp	augmentationDot, accidentalSharp accidentalFlat, accidentalNatural accidentalDoubleFlat accidentalQuarterToneSharpStein accidentalQuarterToneFlatStein accidentalDoubleSharp accidentalThreeQuarterTonesSharpStein	accidental
16th_flag, 8th_flag	flag16thUp, flag16thDown,f lag8thDown, flag8thUp, flag16thUp, flag16thDown, flag8thDown, flag8thUp	flag16thUp, flag16thDown, flag8thDown, flag8thUp, flag32ndUp, flag32ndDown	flag

Table D.2: Filter used to perform the semantic pruning when using the 6_labels granularity. Any edge having a pair of endpoint labels different from those in the list is cut off the candidate graph

Relations types
noteheadBlack - stem,
noteheadBlack - beam,
noteheadBlack - flag,
noteheadBlack - accidental,
noteheadWholeOrHalf - stem,
noteheadWholeOrHalf - accidental