MUSIC COMPOSER RECOGNITION FROM MIDI REPRESENTATION USING DEEP LEARNING AND N-GRAM BASED METHODS

by

Rohit Nitin Kher

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
September 2022

This thesis is dedicated to my family, friends, my pet dog Rani and Professors who supported me in every situation

Table of Contents

Abstract

In order to answer conceptually basic queries like "Who created this piece?" the discipline of computational musicology frequently requires the analysis of detailed characteristics. Melodic lines, rhythmic patterns, chords and chord progressions, tonality, and cadenzas, for example, are all employed. It is feasible to create algorithms that recognise these traits in symbolic data, but it is challenging and takes a lot of expertise. Such an algorithm is significantly more difficult to implement on audio recordings. In the last 10 years, however, machine learning research has enabled various attempts to automatically recreate important audio properties, such as Deep Belief Networks (DBN) and variants of Convolutional Neural Networks (CNN).

In this thesis, I implemented several Deep Learning models and an N-grams model for composer recognition. I used different types of features such as Mel-Spectrograms and Mel-Frequency Cepstral Coefficient (MFCC) with different models such as Resnet, Squeezenet, and Alexnet. My goal was to test alternative approaches to categorising works of Western classical music in order to better understand the composer's most distinguishing characteristics. I created a visual challenge by treating these features as images. I incorporated certain non-traditional methods, such as N-grams for composer recognition, which is methodology adapted from processing natural languages. Some baseline machine learning methods, such as the Naïve Bayes method, are also evaluated. Overall, the best performing method was the Squeezenet model using Mel-Spectograms achieving 93% accuracy for three composers. The N-gram method also provided in some cases high accuracy, such as 93% for three composers.

Acknowledgements

I would want to thank my parents for supporting my higher education and believing in my ambition with all their hearts. I would also like to thank Jennifer Giffin, who has served as a motherly figure during my adventure in Canada, guiding me through many hurdles. Since my arrival in Canada, my pals Samuel Post and David Wauchope have been like family to me. They always inspired me to pursue my thesis and assisted me in coping with personal issues. My colleagues at the Dalhousie Natural Language Processing Lab (DNLP), Stacey Taylor and Tymon Wranik-Lohrenz, assisted me with my thesis from its inception to its final defense. They consistently encouraged and supported me in my efforts.

Lastly, I am grateful to my thesis supervisor, Prof. Vlado Keselj, for his important counsel, unwavering support, and patience. His vast expertise and extensive experience have inspired me throughout my academic studies and everyday life. Since the beginning of my thesis, Prof. Vlado Keselji has always supported my work and experimentation. The timely completion of my thesis would not have been feasible without his ongoing direction and supervision from the thesis's inception through its documentation, mock defenses, and defense. I am really appreciative of the time and work he put into this thesis.

Chapter 1

Introduction

Given the vast increase in digital music data saved and freely available on the cloud, as well as the growing interest in emotive and cognitive computing, identifying composers based on their musical work is an appealing problem for machine learning and artificial intelligence to solve. Few scholars have taken on the automated identification of classical piano composers in the manner that this research suggests. However, many have concentrated on music genre classification to improve recommendation systems and listener experiences. In this paper, I address the efficacy of N-grams on MIDI music scores mixed with rhythmic characteristics for feature extraction, particularly of multi-voice scores. The music library has rapidly expanded, and as it is now publicly available and stored in the cloud, it is being automatically categorised for improved user experience and recommendation systems. The unique style of each classical music composer, which is characterized by a range of components such as the rhythmic structure, which sets the tempo and variations of the piece; the pitch, which produces the melody; and so on, showcases the originality of each composer. Frequently, the lives of the composers or the times in which they lived influenced the development of these distinct genres. My research focuses on discovering the structural characteristics that determine the composer's style in piano works. This increases our understanding of what separates the musical styles of different composers. Although it is often assumed that identifying classical composers needs educated musical hearing, a system that can identify composers from raw data without the requirement for a professional is most desired, as it would help us decide the rightful composer of a piece in case of copyright or ownership disputes.

Furthermore, automating this process becomes more of a necessity than a luxury because it is difficult to remember a style precisely by listening to it, and matching music by reading is even more difficult. In order to address this issue, this study

offers an automated artist identification system built on a special framework for feature synthesis, feature reduction, and supervised classification. In general, a musical composition may be thought of as a recording or a symbolic score. A recording may be made using either an audio file or a MIDI file with real-time sequencing. When music data is preserved as recordings, three types of information are captured: (a) the compositional style of the piece (b) the performer's interpretation of the piece and (c) the piece itself.

Music categorization builds models that give one or more class labels to musical compositions depending on their content. These models are usually assessed by comparing the proportion of properly recognised cases to the total number of occurrences. There are major discrepancies between the data formats utilized by audio data classification methods (such as WAV files) and symbolic data classification techniques (like MIDI files). Diverse types of features may be retrieved from a dataset and utilized to construct models for the two groups. The classification of audio data for music has shown good results. For the purpose of this research, I generated several sorts of features using both audio data (such as WAV files) and symbolic data (such as MIDI files), and then I utilised various Deep Learning and N-grams models to identify the composers.

1.1 Motivation

Music may be categorized by genre, performer, composer, or geographical or cultural origin, to mention a few criteria. A further approach is to organise musical compositions solely based on their self-referential similarities, as opposed to any existing taxonomy. It is not at all easy to immediately categorize music. Since precise, clear, and consistent rules detailing the musical components and qualities of each group are few, categorization based on genre or geography, for instance, may be difficult for both humans and computers. Even creating a taxonomy that is widely accepted might be difficult at times. Similar challenges might arise when developing and using musical similarity measures. When learning about music categorization, whether for academic or practical purposes, these inconsistencies present considerable problems.

Pattern recognition and machine learning approaches enable a wide range of solutions to these problems. These techniques primarily use statistical tools to search

for and learn to recognise empirical regularities in musical collections. The data may be used to develop new taxonomies or linked with existing taxonomies to provide meaning to these statistical regularities. Last but not least, this is used to establish links between the characteristics of musical compositions and interest categories that may be expanded and used to classify new musical compositions.

The applications of automatic music classifiers in academia and business are many. For example, researchers may train classifiers to identify the works of certain composers, then use the learned classifiers to determine if compositions with an unknown authorship are likely to be the work of that composer. They can also use it to determine which characteristics are essential for differentiating across distinct genres and how these attributes vary, genre classifiers may be employed.

As recordings are uploaded to massive databases, a variety of classifiers may be employed to review and categorise them. Depending on the user's interests, music recommendation algorithms may comb through vast music archives of both well-known and obscure music and suggest undiscovered compositions to them. Using classifiers, one may organise large personal music collections according to their owner's mood or the situations in which they would wish to listen to certain songs, such as driving or doing laundry. The identification of unauthorised recordings may be aided by the use of similarity analysis. Certain notes may be automatically classified as various sound occurrences. Genre categorization encompasses the majority of work done in music classification. The acknowledgement of music composers, a more challenging and complex issue, receives less attention.

1.2 Problem Statement

Finding the appropriate dataset was one of the first difficulties encountered throughout this project. Experts in the field may not agree that this dataset represents classical music. Similar arguments might be made about which composers of music belong in the classical category and which ones shouldn't. It was tough from a research standpoint to acquire the proper data since the majority of this music is copyrighted and accessing it is quite difficult. But after considerable searching, I discovered the appropriate dataset on Kaggle titled "Classical Music MIDI".

The given dataset is in the form of MIDI files. The files are in the form of

folders where each folder belongs to a composer. There are total 295 MIDI files. The task here is to find the right features that represent the music in a mathematical or composer-understandable form. At the same time, the task is also to find a set of features that are able to categorise the music distinctly according to composers. The next step or task in line is to find the most optimal Deep Learning or Machine Learning that will classify or categorise the composer with high accuracy. The final step in this would be to find the ideal combination of features and models such that the maximum accuracy is achieved. Apart from this, I am also implementing the N-grams algorithm. So the task here is to make N-grams, which are mostly used as features in the N-grams model, in a way that gives a high level of accuracy.

I have explained the MIDI and features such as Mel-Spectrogram and MFCCs in the background section. I have explained the different deep learning models and N-grams in the methodology section.

Chapter 2

Related Work and Background

2.1 Related Work

The problem of composer classification has gotten less attention than computerized music categorization, which has attracted a lot of study. Music categorization builds models that give one or more class labels to musical compositions depending on their content. These models are usually assessed by comparing the proportion of properly recognised cases to the total number of occurrences.

Hontanilla *et al.* [8] provides a language modelling tool that uses N-grams to represent different composers' styles. They created N-grams using pitch intervals and inter-onset duration ratios (IOR) of consecutive notes. To replicate the experiments done in earlier writings by other authors, they employed a corpus of five composers from the Baroque and Classical eras. In these experiments, they found evidence that the results may be influenced by factors other than the composers' personal preferences, such as the diversity of the musical genres selected for the corpus. A fresh experiment was run using a corpus of fugues by Bach and Shostakovich to assess the efficacy of modeling techniques in emulating the individual stylistic characteristics of the artists. According to the confusion matrices, all of Bach's works were correctly labeled, and only 2 of Shostakovich's works were wrongly labeled as Bach's. These great results show that the language models were able to tell these two composers apart, even though they used the same musical forms. They made an effort to emulate the styles of several musical composers using language modelling techniques. To show the effectiveness of N-gram models for this job, they repeated the research using the same corpus of composers. According to the results of these investigations and an analysis of the corpus's composition, it is quite challenging to fully pinpoint a composer's particular style. For the chosen musical works in their examinations, they used digital scores. In these compositions, the composers used a musical language characterized by their own styles, which are influenced by a number of outside

factors such as the composer's temporal context, the piece's genre, or the musical form. In a recent experiment employing a corpus of Bach and Shostakovich fugues, they were almost able to extract the distinctive features of each artist by applying language models. In this experiment, just one musical form was used, and they tried to minimize the factors affecting the composer's styles such that only their unique characteristics separated the datasets. The language models were effective in accurately reflecting each composer's style, as seen by their high success rate of 96.6%. In any event, even when merging different research, their modelling technique has shown to be adaptable enough to capture the attributes of each dataset. They came to the conclusion that musical styles may be represented using language models as a consequence.

Micchi [16] describes a neural network strategy for extracting musical elements from 20-second audio recordings and predicting the composer. A long short-term memory recurrent layer comes after three convolutional layers in the network. The model is 70% accurate on the validation set when distinguishing among six composers. The research is the first phase of a project with automated feature detection and visualisation as its main goals. They presented a neural network topology that consists of a CNN connected to an RNN for automated composer classification.

Jain *et al.* [11] experiment with several techniques of classifying works of Western classical music by composer to better understand the composer's most distinguishing characteristics. They decided to use the silent MIDI musical score encodings. When comparing the performances of 15 different composers from various historical eras, decision tree boosting fared better than all other models (79% test accuracy). For classification, vertical intervals and texture were considerably more useful than rhythmic components. They created a visual challenge by translating audio snippets into visual representations that were put into a deep CNN. Using jSymbolic, they converted each MIDI file into an image of size 64×1536 pixels. Most of the notes in our pieces are between notes 32 and 96, which is why the vertical axis of the image is made up of 64 different MIDI notes. Time is on the horizontal axis. With a sampling rate of 10 Hz, the first 153.6 s of the MIDI file were used to make an image that is 1536 pixels wide. Six composers were surprisingly able to reach a test accuracy

of 70%. Gradient boosting with decision stumps performed better than a conventional fully-connected NN in classifying 15 composers, with an accuracy of 79.1%. The binary classification experiment using logistic regression revealed that composers from comparable eras were more difficult to distinguish than those from obviously separate eras. Comparing composers who composed music for different instruments revealed that melodic and vertical intervals were far more unique than rhythm, and instrumentation was a dead giveaway. Considering the vast amount of information lost while converting a MIDI file to a score representation, the 70% accuracy on six composers was remarkable.

Wolkowicz *et al.* [20] explains how Natural Language Processing and Information Retrieval technologies can be used for music. A method for converting complicated musical structures into characteristics (N-grams) that correlate to text terms has been developed. The challenge of automatic composer attribution was resolved by adding case-specific theoretical features to the well-known statistical NLP approach of statistical analysis of N-gram profiles. The data was extracted from a corpus of piano music MIDI files. This study reveals several facets of the problem and demonstrates that music can be managed using NLP and IR technologies. After demonstrating that some natural language processing techniques are applicable to music, individuals may seek to build further methods, such as clustering, plagiarism detection, and music information retrieval systems, among others. Using N-gram interpretation, it may be possible to index and explore musical collections more effectively.

Abeler [1] states that identifying musical composers is the most difficult task in music information retrieval. It is necessary to use music notation or recordings to identify the composer of a musical work. Since the piece in question is unidentified, the choice must be made using the composer's earlier work.

Herremans *et al.* [6] examines two problems: first, how to develop an autonomous system that can recognize music by various composers; and second, how to determine the key musical properties for this task. They built five distinct classification models utilizing a data-driven methodology and a vast corpus of existing music in order to properly distinguish between three composers (Bach, Haydn and Beethoven). Black-box models such as support vector machines are developed in conjunction with more intelligible models such as decision trees and rule sets. The researchers said that the

first three models—C4.5 decision tree, RIPPER ruleset, and logistic regression—are easier to understand and give more information about how each composer's style is different and what makes it unique. While the last two models, Naive Bayes and Support Vector Machines, are complicated to interpret, they are used as performance benchmarks. For this research, a diverse selection of three composers' works was mined for a variety of universal musical properties (Bach, Beethoven and Haydn). On the basis of these characteristics, five categorization models were built. The first three models give a deeper grasp of the qualities and differences of each composer's style. Due to their complexity or size, the latter two models are used as performance benchmarks. Even while black-box models (SVM) provide the best results (AUC of 93% and accuracy of 86%), comprehensible models such as the RIPPER ruleset nevertheless perform well (AUC 85% and accuracy 81%). The basic models provide musical insights and may guide our future musicology studies in the right direction. The findings of this research indicate that, unlike Haydn or Bach, Beethoven does not generally emphasize a single interval. This conclusion cannot be generalized without more research since it is evidently based on a small corpus.

McKay *et al.* [14] empirically proves the efficacy of features based on instrumentation in the field of automated music categorization. The value of high-level features generally, as opposed to low-level signal-processing-based characteristics, is shown through an experiment involving computerised genre classification. Experiments also permit the use of large feature sets and feature weighting techniques. In this research, tools are described together with pertinent background data. The overview provides a comprehensive list of a broad range of pertinent musical features that are open to study for theorists, musicologists, and other academics. For academics interested in classifying music based on instrumentation and other high-level qualities, the Bodhidharma symbolic music classification system is also offered as a useful and straightforward resource. The Bodhidharma System consists of 111 musical features, all of which can be extracted from MIDI files. These features can be divided into 6 subgroups, such as instrumentation, rhythm, texture, dynamics, pitch statistics, and melody. According to the research described in this article, high-level qualities may be very helpful in categorising music. Given that a large portion of earlier research on computerised music classification has focused on low-level characteristics,

this is an important conclusion. In an experiment using just high-level features, MIDI recordings were accurately categorised into 9 categories with 90% accuracy and 38 categories with 57% accuracy. These success rates are far higher than those of earlier experiments using both symbolic and aural data. This research demonstrated that a large library of features may significantly outperform smaller libraries when used in combination with feature weighting methodologies and hierarchical and/or round-robin classification. A number of criteria were assessed for overall effectiveness, and it was found that although a small number (7 out of 111) had a dominating impact in several classification categories, even very minor aspects were crucial. The qualities based on instrumentation (i.e., timbre) were shown to be by far the most crucial category of features. Instrumentation was comprised of 20 features with a combined weight of 41.8% across six feature groups. In addition to receiving over twice as much weighting as the second-best performing feature group, instrumentation-based features also made up two of the top three individual features. The relevance of instrumentation implies the significance of timbre generally, since the human perception of instrumentation while listening to audio signals is primarily an abstraction drawn from timbral information. This paper's findings support the recommendation that instrumentation should get particular consideration in future theoretical and practical investigation and analysis. This is an interesting result since instrumentation is typically placed in a supporting role in the literature. These findings likewise imply that the development of instrument identification sub-systems for audio classification systems should be prioritised as the high-level instrumentation features that result could be more profitable than the low-level timbral features currently employed in audio classification.

Maximos *et al.* [12] states that many attempts to retrieve music information using computational intelligence approaches have been done in the recent decade. A study was conducted to explore the informative ability of the Dodecaphonic Trace Vector for the purpose of composer identification and classification. Using probabilistic neural networks, they created a "similarity matrix" of different composers, and then evaluated the Dodecaphonic Trace Vector's capacity to recognize a composer using trained feedforward neural networks. Dodecaphonic Trace Vector is created by normalizing 12-dimensional Chroma profiles. The training approach employs both standard

gradient-based techniques and the Differential Evolution algorithm. Seven classical composers' works are analyzed experimentally in order to comprehend the approach's primary advantages and limits. This research demonstrates that the Chroma Profile Vector (based on which Dodecaphonic Trace Vector were created) contain a wealth of information about the personality of a composer. In addition, professionals with broad musical backgrounds must assist the musical analysis with important data and contribute to the research results.

Hajj *et al.* [3] states that given the enormous expansion of digital music data stored and freely available in the cloud, as well as the rising interest in emotional and cognitive computing, it is an attractive topic for machine learning and artificial intelligence to examine whether or not it is possible to identify composers based on the musical compositions they have produced. In this study, they examine the use of N-grams for extracting features from MIDI music scores, including rhythmic qualities, focusing on multi-voice scores. In addition, cortical algorithms (CA) are enhanced to successfully identify composers in a supervised way while simultaneously decreasing the enormous feature set generated. Their proposed technique produced a 94.4% identification rate on a database of 1197 pieces manufactured at home using just 0.1% of the 231,542 generated characteristics, prompting more study. In actuality, the preservation of the most essential traits yielded remarkable outcomes in terms of duplicating the musical styles of keyboard composers. Using a new N-gram-based approach for feature extraction and a cortex algorithm for feature reduction and recognition phases, they constructed an automatic identification framework for recognizing classical music composers. Their recommended method is effective due to the robustness of CA-based feature reduction and the discriminative capability of their proposed feature. Only 0.1% of the 231542 features discovered by CA were employed in their proposed technique to obtain a 94.4% identification rate on 1197 compositions by 9 composers, which promotes research on bigger datasets.

Yang *et al.* [21] looks at how composer styles are used to classify sheet music, MIDI, and audio data for the piano. In three ways, the researchers add to what has already been done. They try out different ways to add to data that are inspired by music, such as changing the pitch of notes or removing notes or groups of notes at random. They show that these augmentation schemes improve model performance

in a way that is bigger than the benefit of pretraining on all images of solo piano sheet music in IMSLP. Second, they explain how to change previous models to make cross-model transfer learning possible. This is a way that a model trained only on sheet music can be used to classify composers from audio or MIDI data. Third, they look at how well trained models work in a 1-shot learning context, where the model has to classify a set of composers it has never seen before. Their results show that models learn a representation of style that works for more composers than just the ones they were trained on.

Verma *et al.* [19] in his paper looks at models that can be used to figure out who wrote a piece of music. The authors show how to do this task with several pooled, convolutional architectures and show how their method is similar to traditional learning methods based on global and n-gram features. Authors test models on 2,500 scores from the KernScores collection. These scores were written by a wide range of composers from the Renaissance to the early 20th century. This corpus has a lot in common with the corpora used in a number of smaller studies that came before it. Authors compare their results on subsets of the corpus to those of these smaller studies. Overall, the authors conclude that the convolutional models suggested in their paper do a pretty good job. Also, the authors don't think that the full potential of these methods has been reached. More research could lead to even better convolutional architectures for classifying composers.

Tsai *et al.* [18] look at how composer styles can be used to classify pictures of piano sheet music. Before, it was hard to classify composers because there wasn't enough information. Authors deal with this problem in two ways. First, they alter the issue such that it is based on unprocessed photographs of sheet music as opposed to a symbolic music format. Second, they propose a technique that can be trained on unlabeled data. Their technique first converts the sheet music picture into a string of musical "words," which are then sent into a text classifier, based on the bootleg feature representation. The authors demonstrate that by first training a language model on a collection of unlabeled data, beginning the classifier with the weights from the pretrained language model, and then fine-tuning the classifier on a tiny quantity of labeled data, the performance of a classifier may be significantly improved. The AWD-LSTM, GPT-2, and RoBERTa language models are all trained using the

piano sheet music pictures in IMSLP. According to the authors, transformer-based architectures outperform CNN and LSTM models, and pretraining boosts the GPT-2 model's classification accuracy on a 9-way classification test from 46% to 70%. The projected piano score may also be used to project the compositional style onto a feature space using the trained model. The authors show that the performance of the classifier may be significantly enhanced by first training a language model on a large amount of unlabeled data.

2.2 Background

In this section, I will provide some background information regarding symbolic representation of music such as MIDI, different types of features incorporated such as Mel-Spectrograms and Mel-frequency Cepstral Coefficients, and the N-grams model used for composer recognition.

2.2.1 MIDI

Musicians may play, edit, and record music using a wide variety of electronic musical instruments, computers, and associated audio equipment when using the MIDI (Musical Instrument Digital Interface) communications protocol, digital interface, and electrical connections. Up to sixteen MIDI channels, each of which may be addressed to a separate device, can be transported via a single MIDI connection. Each time a key, button, knob, or slider is moved, a MIDI event that includes musical instructions such as note pitch and velocity is created. A common MIDI use involves using a MIDI keyboard or other controller to generate noises that the audience hears via a keyboard amplifier by activating a digital sound module (which contains synthetic musical sounds). MIDI data may be recorded and altered using a sequencer or digital audio workstation, or it can be transferred through a MIDI or USB link. For the purpose of storing and transferring data, this file format was created. The benefits of MIDI include its small file sizes, ease of modification and manipulation, and wide variety of electronic instruments, synthesisers, and digitally sampled sounds. However, because MIDI records messages and information about notes rather than specific sounds, this recording could be altered to sound like anything from a synthesised or

sampled guitar or flute to a full orchestra. A MIDI recording of a keyboard performance may resemble the sound of a piano or other keyboard instrument. A MIDI recording is not an audio transmission, as opposed to a microphone-recorded sound recording.

MIDI events may be sequenced using computer software or specialised hardware music workstations. A lot of digital audio workstations (DAWs) are designed from the ground up to use MIDI as a primary feature. To make it simple to edit recorded MIDI messages, several DAWs now include MIDI piano rolls. These technologies allow composers to audition and modify their work far more rapidly and effectively than they could with older methods like multitrack recording. Because MIDI is a collection of sound-generating instructions, MIDI sequences may be modified in ways that cannot be done with prerecorded audio. Changes may be made to the key, instrumentation, tempo, and component part order of a MIDI arrangement. By creating ideas and hearing them quickly by playing them back, composers may experiment. Algorithmic composition software is used to make computer-generated performances, which can be used as song inspiration or as an accompaniment.

2.2.2 Mel-Spectrograms

A signal's time-varying frequency spectrum may be seen in a spectrogram, which is a visual depiction of the concept. Spectrograms are utilized in numerous disciplines, including music, linguistics, sonar, radar, speech processing, and seismology. Audio spectrograms can be utilized to analyze different animal sounds and phonetically identify spoken words.

Human hearing interprets the tones on the mel scale as being evenly spaced apart. As frequency increases, the Hertz gap between mel scale values (also known as "mels") widens. The scale is based on pitch comparisons, as indicated by the term mel, which derives from the word melody. Hertz data is transformed into mel scale values using the mel spectrogram. When modelling human hearing perception, mel spectrograms are preferred over linear audio spectrograms because they are better for applications where all frequencies are equally essential. Mel spectrogram data may also be useful for audio categorization programs.

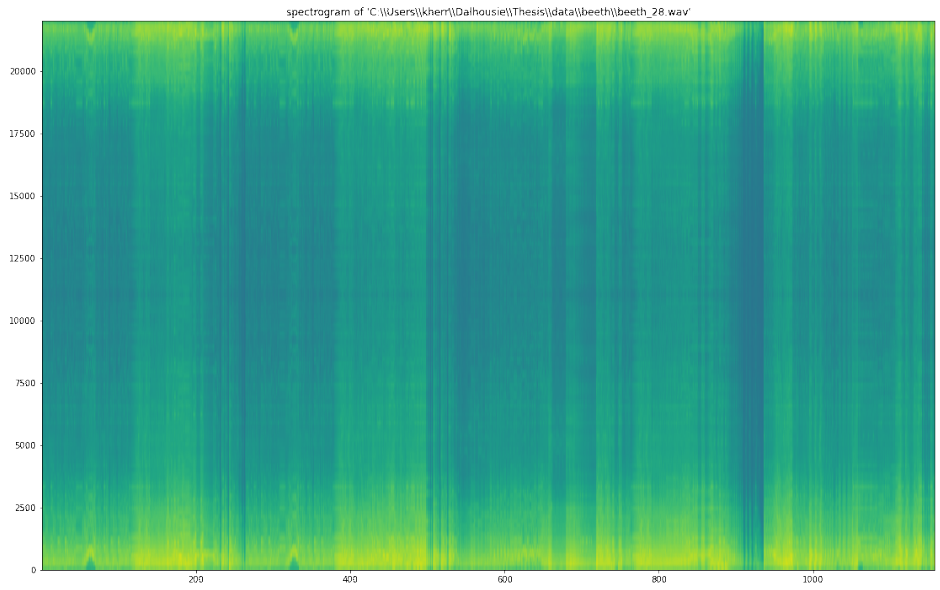

Figure 2.1: Mel-Spectrogram

2.2.3 Mel-frequency Cepstral Coefficients

A depiction of the short-term power spectrum of a sound is called the mel-frequency cepstrum, which is more often abbreviated as MFC. Cepstral is calculated using Inverse Fourier Transform (IFT) of logarithm of signal spectrum. It converts a signal from time domain to frequency domain. The MFC is useful in the realm of sound processing in a number of applications. Mel-frequency cepstral coefficients, or just MFCs, are the coefficients that are used to make an MFC. They are produced by using the cepstral representation of the audio sample in the generation process. The linearly spaced frequency bands of the normal spectrum are what differentiate the mel scale from the linearly spaced frequency bands of the normal spectrum. This is because the mel scale more accurately mimics the response of the human auditory system. For instance, warping the frequency of the sound may help improve the audio compression process so that it more accurately depicts the sound.

The process for creating MFCCs is typically as follows:

1. Apply the Fourier transform to a signal (in a windowed excerpt).

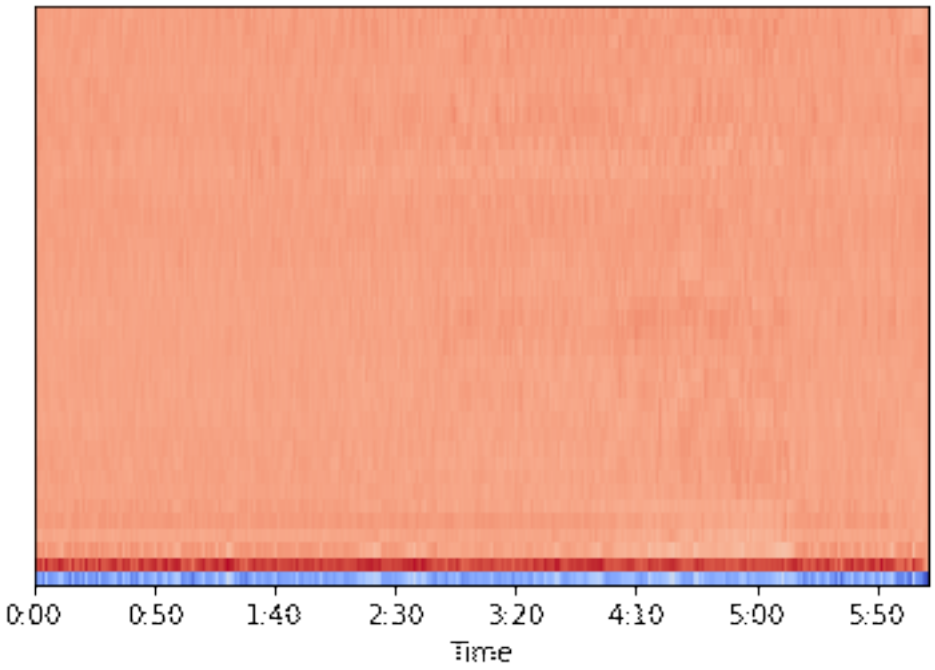

Figure 2.2: Mel-frequency Cepstral Coefficients

2. To translate the spectral powers from the previous section onto the mel scale, use triangle overlapping windows or cosine overlapping windows.

3. Examine the power records for each mel frequency.

4. Determine the discrete cosine transform of the mel log powers as a signal by doing the calculation.

5. The resulting spectrum's amplitudes are the MFCCs.

Alterations to this strategy include adding dynamics features such as "delta" and "delta-delta" coefficients, which measure the first- and second-order frame-to-frame difference, as well as modifying the form or spacing of the windows that are used to map the scale. These coefficients measure the difference between consecutive frames in first and second order, respectively. They are often used in speech recognition systems as features, such as those that can recognise spoken phone numbers. Applications for music information retrieval that employ MFCCs include genre classification, aural

similarity tests, and others that are growing quickly. Speech recognition systems often normalise MFCC values to lessen the influence of additive noise since they aren't especially stable in the presence of noise. Some studies suggest changing the fundamental MFCC approach to improve robustness. For instance, before executing the discrete cosine transform (DCT), which lessens the impact of low-energy components, boost the log-mel-amplitudes to an acceptable level.

2.2.4 N-grams

An N-gram, which may also be referred to as a Q-gram, is a continuous sequence of N components taken from a specific text or audio sample. In the domains of computational linguistics and probability, N-grams are a common data structure to work with. Depending on the setting in which they are being used, the components may take the form of phonemes, syllables, letters, words, or base pairs. The extraction of N-grams from a text or audio corpus is a standard data mining technique. A "unigram" is a size 1 N-gram, a "bigram" is a size 2 N-gram (or, less often, a "digram"), and a "trigram" is a size 3 N-gram. It is a kind of probabilistic language model that takes the form of a $(n-1)$-order Markov model, and it is used to predict the next element in a sequence.

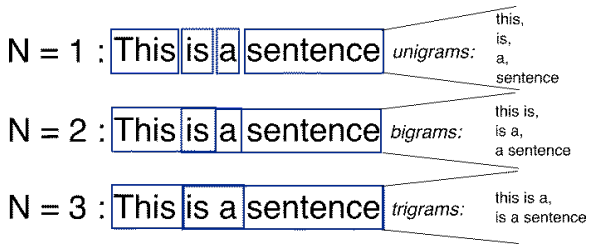

Figure 2.3: N-grams

N-gram models are widely used in the academic disciplines of probability, communication theory, statistical natural language processing, biological sequence analysis, and data compression. The N-gram models and the algorithms that use them have a number of benefits, such as being easy to use and their capacity to scale. When the value of N is increased, a model has the potential to save more context in a way that is both space and time-efficient. N-gram models are quite commonplace in the field of statistical natural language processing. In speech recognition, a representation

of phonemes and sequences of phonemes is accomplished via the use of an N-gram distribution. In order to facilitate parsing, each word is encoded in such a way that its N-gram will consist of exactly N words. The sequences of characters in different languages, like the letters of the alphabet, are modeled to make it easier to tell which language is being used.

In addition to word sequences, N-grams may be used to represent almost any kind of data. They have been used, for instance, to extract traits for classifying vast quantities of satellite earth photographs and identifying the location of a specific image on the globe. As a first step, they have also had great success in finding genetic sequences and determining the species from which short DNA sequences originated. Another criticism is that the performance/competence dichotomy is not accurately reflected by Markov language models, such as N-gram models. This is because N-gram models are used in practical applications rather than as models of linguistic knowledge, since they don't claim to be (even in theory) complete representations of linguistic information.

Chapter 3

Methodology

3.1 Dataset

One of the significant challenges was locating the appropriate dataset. For different deep learning and N-grams models to work, the music files in the data set must also be in MIDI format. This is because N-grams need information about the pitch and length of notes. After doing some investigation, I was able to locate an adequate dataset on Kaggle. Google LLC operates the online data science and machine learning community known as Kaggle. Users are able to find and upload data sets; communicate with other data scientists and machine learning specialists; explore and develop models in a web-based data science environment; and participate in data science contests via Kaggle.

For this research, I would be using a Kaggle dataset titled "Classical Music MIDI" which consists of 19 classical composers. This dataset is structured in the form of folders containing MIDI files for each composer. I would like to clarify that even though the dataset is named as "Classical Music MIDI" there can be difference of opinion among music enthusiast about whether a composer belonged to classical music or not. When I use the term "classical", I often mean what a layperson would define as classical music, which is an older kind of vintage music without vocals and mostly piano or other instrumental accompaniment. Since it meets the parameters for the dataset needed for this study, I have only taken this dataset into consideration.

3.2 Methodology Approach

After examining the dataset, I came to the conclusion that it was not balanced. The number of MIDI files for each composer differed greatly. For instance, the Bach composer in the dataset only has 3 files, but Chopin has 48 files. Unbalanced datasets are an issue for two main reasons:

1. Because the model or algorithm is never given enough time to consider the underlying class, it does not provide the best outcomes in real time for the unbalanced class.

2. When the number of observations for a few classes is very low, it is challenging to have representation across classes, which makes it difficult to create a validation or test sample.

There are three options offered in case of unbalanced dataset, each having advantages and disadvantages of its own:

1. If a class has enough observations for the comparison ratio between two classes, delete it at random. Even though this method is pretty easy, there is a good chance that the data we are getting rid of contains important information about the expected class.

2. Increase the number of observations in the unbalanced class at random, basically creating duplicate samples. This should provide us enough samples to conduct our experiments with. The training data may be overfit due to oversampling.

3. The synthetic production of unbalanced class observations that are equivalent to the present nearest neighbours classification is required by the synthetic sampling (SMOTE) technique. What to do when there are few observations of a highly uncommon class is the problem.

Because the number of files for each composer varied greatly, I simply excluded a set number of composers. The length of works by different composers varied. But if we cut all the works down to the same length, we would lose a lot of information and features; and, additionally, all the works would be turned into visual representations of the same size before feeding them to the models. In the end, I decided to use 11 composers and the same number of files for each composer. In a similar way, the quantity of classes also influences how accurate Deep Learning models are. The easiest way to figure out what accuracy "means" in terms of the number of classes is to look at a random baseline. A coin toss has an accuracy of 1/K, where K is the total number of classes. Therefore, 50% for two groups, 10% for ten, and 1% for one hundred. This indicates that when there are more classes, accuracy of "60%" means

more. A binary classifier with an accuracy of 60% is close to random, but 60% for 100 classes shows strong classifier performance. Since composer identification is a highly challenging topic, I had far less data than is often used to train deep learning models. I made the decision to run my Deep Learning models on various numbers of labels in order to thoroughly comprehend my findings. I used 11, 5, and 3 labels or classes to run the deep learning models. The idea behind this was to understand the effect of the number of composers, the number of examples, and the behavior of different composers across different features and models. I tried to give the maximum number of examples in each case. I allocated 10 files for training in the case of 11 composers, i.e., 110 files in total. In the case of 5 composers, I allocated 20 files for each composer, i.e., 100 files in total, and in the case of 3 composers, I allocated 33 files for each composer, i.e., 99 files in total.

Feature extraction is a vital step in investigating and establishing links between various objects. Feature extraction is utilized to convert the auditory data into a format that can be comprehended since the models cannot directly comprehend the auditory input. An approach that displays the majority of information in a straightforward manner. A necessity for all classification, prediction, and recommendation systems is feature extraction. Audio features—descriptions of sounds or audio signals—can be included into statistical or machine learning models for the development of intelligent audio systems. These features are used in several audio applications, including audio classification, voice recognition, automated music tagging, audio segmentation and source separation, audio fingerprinting, audio denoising, and music information retrieval. Different characteristics capture various components of sound. General audio feature categories include the following:

1. Levels of abstraction are used to divide musical signals into high-, mid-, and low-level parts. Rather than covering audio in general, these broad categories primarily encompass musical signals:

    (a) High level: These are the abstract elements that people can understand and value. Examples include the instrumentation, key, chords, melody, harmony, rhythm, genre, and atmosphere.

   (b) Mid-level: These are traits we could see. Examples include pitch, beat-related traits, note onsets, fluctuation patterns, MFCCs, and more. These may be thought of as a collection of base attributes.

   (c) Low-level traits: These low-level characteristics were derived from audio. Machines can comprehend them, but humans cannot. Some of these are amplitude envelope, energy, spectral centroid, spectral flux, zero-crossing rate, and others.

2. Temporal Scope: Local, global, and instantaneous time-domain features. All audio, whether musical and non-musical, falls under this classification:

   (a) Instantaneous: As the name suggests, these traits provide us with quick information about the audio stream. These consider millisecond-sized chunks of the audio stream. Humans can distinguish a minimum temporal resolution of around 10 milliseconds.

   (b) Segment-level: These properties may be ascertained from an audio stream's seconds-long segments.

   (c) Global features: These are a group of characteristics that describe and provide information about the whole sound.

3. Musical Aspect: Pitch, harmony, melody, beat, rhythm, timbre (sound colour), and other acoustic characteristics.

4. Signal Domain: Features in the time, frequency, or both domains make up the signal domain. The most important or descriptive parts of audio in general are in the signal domain:

   (a) Time domain: These features are obtained from the unprocessed audio waveforms in the time domain. Examples include RMS energy, amplitude envelope, and zero crossing rate. They can be explained as follows:

      i. The signal's amplitude envelope is composed of all samples' maximum amplitude values for each frame. It gives a rough idea of how loud something is with this tool. However, it is vulnerable to abnormalities. This trait has been used in a lot of different ways, especially for finding the start of a sound and sorting music into different genres.

ii. Root Mean Square Energy is calculated using all the data within a frame. It acts as a volume indicator since louder sound is produced at greater energies. It is more resistant to outliers than the Amplitude Envelope. It has been shown that this capacity is useful for tasks like audio segmentation and music genre classification.

iii. The zero-crossing rate of a waveform is the frequency at which it crosses the horizontal time axis. This feature has mostly been used in applications such as voice/unvoiced speech signal judgement, monophonic pitch estimations, percussive vs. pitched sound detection, and other applications.

(b) Frequency domain: These features concentrate on the frequency elements of the audio stream. The Fourier transform converts signals from the time domain to the frequency domain. The band energy ratio, the spectral centroid, and the spectral flux are some examples.

(c) Time-frequency representation: This function combines the time and frequency components of the audio input. The time-frequency representation may be created by applying the Short-Time Fourier Transform, often known as the STFT, to the time domain waveform. The spectrogram, mel-spectrogram, and constant-Q transform are a few examples.

5. We can distinguish two methods for selecting features in machine learning: manually selecting features for traditional ML modelling and automatically extracting features for deep learning modelling. Let's look at them in more detail:

(a) In a typical machine learning approach, all or the majority of the features from both the time and frequency domains are inputs into the model. Feature selection must be done manually based on how they affect model performance. Some of the most frequently used features are Amplitude Envelope, Zero-Crossing Rate (ZCR), Root Mean Square (RMS) Energy, Spectral Centroid, Band Energy Ratio, and Spectral Bandwidth.

(b) The Deep Learning approach takes into consideration unstructured audio representations like the spectrogram or MFCCs. It has the ability to discover patterns on its own. The enormous quantity of data and computer

power readily accessible also helps.

Mel-Spectrograms, Mel-Frequency Cepstral Coefficients (MFCC), 12 audio characteristics retrieved from the jSymbolic program, and N-grams are some of the features used in this study. Choosing the appropriate neural network to do composer recognition is a crucial next step.

The term "neural network" may either refer to a network or circuit that is made up of artificial neurons or nodes, which we normally assume in Computer Science, or it may refer to a network that is made up of biological neurons. An artificial neural network can be made up of artificial neurons or nodes. The problems that need to be solved by artificial intelligence (AI) are often approached using artificial neural networks (made up of artificial neurons), which mimic how actual biological neural networks (which are made up of biological neurons) work. Weights between nodes in an artificial neural network indicate the connections that would exist between biological neurons in a natural neural network. An excitatory connection receives a weight that is positive, whereas an inhibitory link receives a weight that is negative. Each individual contribution is given a weight before the total is calculated. A linear combination is the name given to this particular technique. In conclusion, the amplitude of the output is under the control of an activation function. These artificial networks are used for applications needing training data, such as adaptive control and predictive modeling. Within networks that are able to derive conclusions from a convoluted and apparently disconnected collection of data, there is the potential for self-learning that is based on experience to take place. The approach known as neural networks was developed with the intention of solving issues in a way that was analogous to that of the brain of a human being. Backpropagation entails the transmission of information in the opposite direction and the modification of the network to reflect this information. Neural networks can be used for many different things, like computer vision, voice recognition, machine translation, filtering information in social networks, playing board games and video games, and making medical diagnoses.

Deep neural networks are a kind of artificial neural network that is characterized by the presence of several degrees of connectivity between the input and output layers of the network (DNN). Neurons, synapses, weights, biases, and functions are the fundamental building blocks that are necessary for the construction of neural

networks of any size or shape. For instance, a DNN that has been taught to identify different breeds of dogs may look at the picture that has been supplied and determine the chance that the dog presented is of a certain breed. The user may do an analysis of the data and decide which probability the network should expose before making a suggestion for a label (those that are higher than, say, a certain threshold). The term "deep" networks refers to DNNs that have numerous layers, where a layer consists of neurons at one level making one step of analysis of a set of signals. DNNs are able to accurately describe intricate nonlinear interactions because of their modeling capabilities. When building compositional models using DNN architecture, the item being modeled is described as a layered composition of primitives. These models are built using compositional modeling. Because of the additional layers, it is now able to combine qualities from lower levels, which may make it possible to simulate complex data with a smaller number of units than is possible with a shallow network that performs in an equal manner. For example, it has been shown that when working with sparse multivariate polynomials, DNNs are much more accurate in estimating than shallow networks. The first thing that the DNN does is create a map of virtual neurons and then assign random numerical values, sometimes known as "weights," to the connections that are made between those neurons. When the inputs are multiplied by the weights, a value that falls between 0 and 1 may be obtained. The weights would be adjusted by an algorithm in the event that the network was unable to correctly recognize a pattern. It is possible that some factors will have a greater influence than others until the algorithm learns to completely grasp the data via the use of mathematical manipulation. Due to the fact that input may come from either the past or the present, recurrent neural networks, also known as RNNs, are used in applications such as language modeling. For this purpose, having a memory for the short term is particularly useful. In the area of computer vision, the use of convolutional deep neural networks, more often referred to as CNNs, is common. Additionally, CNNs have been used in the field of acoustic modeling for the purpose of automated speech recognition (ASR). For the purpose of this research, I would make use of Resnet, Alexnet, Squeezenet, CNN, and LSTMS with a total of 34 layers. These different deep learning models and N-grams methodology will be explained in detail in the upcoming sections.

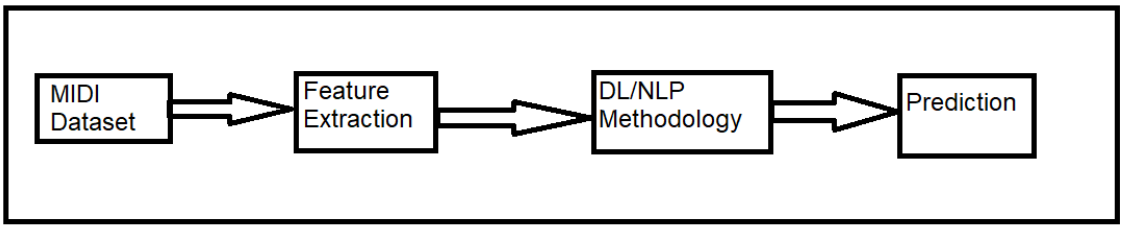

Figure 3.1: Methodology

The approach in my study, shown in the Figure 3.1, can be summarised as follows:

1. Data preparation: This involves converting MIDI files to WAV files for feature extraction.

2. Feature Extraction:

   (a) Extracting Mel-Spectrograms and MFCCs from the WAV files for implementing Deep Learning models

   (b) Extraction of the pitch and duration of notes from MIDI files for generating N-grams

   (c) Extraction of 12 audio features using the jSymbolic software.

3. Utilizing several Deep Learning models and N-grams Model: It involves implementing different models such as Resnet, Squeezenet, Alexnet, CNN, and LSTM on Mel-Spectrograms and MFCCs. Implementing N-grams models using the N-grams algorithm by creating profiles using the N-grams. Implementing Naïve Bayes using the 12 audio features.

4. Model optimization to get the greatest outcomes: This involves changing the hyperparameters in the case of deep learning models and the value of N and profile lengths in the case of N-grams.

5. Prediction

3.3 Feature Engineering

The process of selecting, modifying, and transforming raw data into features for the sake of supervised learning is referred to as feature engineering. It's possible that for

machine learning to work well with new tasks, more efficient features will need to be built and trained. Any measurable input that might potentially be employed in a predictive model is referred to as a "feature." It might be the shade of an item or the inflection of a person's voice. When it comes to working with models for machine learning, feature engineering has a very important function. A flawed feature will have an immediate effect on your model, regardless of the data or the way it was designed. Eliminating features from a dataset that are not useful is a part of the process known as feature extraction, which is done so that usable information may be extracted. This can be done without affecting the original connections or the critical data, and it brings the total quantity of data down to a level that is manageable for algorithmic processes.

One of the first steps in feature engineering is the extraction of mel-spectrograms and mel-frequency cepstral coefficients (MFCC) from MIDI files.

But MIDI files have to be changed into WAV files first because they can't be changed directly to MFCCS or Mel-Spectrograms.

IBM and Microsoft came up with the WAV file format in order to store audio that has not been manipulated in any way. The containers used by this format are responsible for storing audio data together with track numbers, sample rates, and bit rates. WAV files have a maximum file size limit. Audio data is sent using Resource Interchange File Format (WAV) containers in "chunks" that are not processed and are often not compressed (RIFF). Windows often makes use of this method for the storage of audio and video files such as AVI, despite the fact that it may also be used to store random data. Because WAV files are often uncompressed, their size is much greater compared to that of other prevalent audio file formats such as MP3 (compression is supported, though). In order to maintain the finest possible audio quality, they are thus employed almost exclusively in the professional music recording industry. I made use of third-party software in order to convert MIDI files to the WAV format. Using the 'Librosa' library, I was able to produce Mel-Spectrograms.

Librosa is a Python software library that can analyze music and sound. Librosa is a program that is often used for dealing with audio information, such as in automated speech recognition and the production of music (using LSTMs). It addresses

the fundamental elements that must be present prior to the construction of music information retrieval systems. For the purpose of assisting in the extraction of features and the presentation of audio sources, Librosa makes use of a variety of different signal processing methods. I used Librosa to extract mel-spectrograms and mel-frequency cepstral coefficients (MFCC), and then I saved those results as PNG picture files and placed them in composer-specific folders.

3.3.1 Mel Spectrograms

The Quick Fourier transform is an efficient method for examining the frequency content of a signal. Running the FFT on overlapping windowed portions of the stream creates the spectrogram. One way to conceptualize a spectrogram is as a series of FFTs stacked on top of one another. It is a graphic depiction of a signal's loudness, or amplitude, as it varies over time and at various frequencies. There are various other processes taking place in the background while constructing the spectrogram. Decibels are used to represent the color dimension, and a log scale is used for the y-axis (you can think of this as the log scale of the amplitude). Mel Spectrograms have been explained in previous sections.

3.3.2 Mel frequency Cepstral Coefficients

When dealing with audio signals, Mel frequency Cepstral Coefficients (MFCC) technique is commonly used since it is one of the most crucial ones for extracting a feature from an audio signal. The Mel frequency of a signal Cepstral Coefficients (MFCCs) are a condensed set of properties (typically 10–20) that accurately describe the general shape of a spectral envelope. Using Mel Frequency Cepstral Coefficients (MFCCs), audio properties may be extracted. The MFCC separates the frequency band into sub-bands using the MEL scale, and then uses the Discrete Cosine Transform to extract the Cepstral Coefficients (DCT). Sound processing is made easy by the MEL scale, which is based on how individuals distinguish different frequencies. The Mel frequency Cepstral Coefficients have been explained in the previous sections.

3.3.3 12 audio features using jSymbolic

I used machine learning models to analyze the results of the twelve audio features extracted using jSymbolic. The piece of software known as jSymbolic was developed with the intention of assisting scholars working in the fields of music information retrieval (MIR), musicology, and music theory. The major objective of this program is to decipher musical data that has been symbolically encoded in file formats such as MIDI and MEI in order to derive statistical information. This statistical data is represented as feature values, and it can be used to query massive musical databases, feed directly into automatic classification systems, or perform empirical musical studies by musicologists and theorists. All of these methods are possible thanks to the data's representation as feature values. The feature library that is included with jSymbolic has a total of 246 unique features. Some of these features are multidimensional, and the total number of these values is 1497. These characteristics are the result of a comprehensive investigation of a wide range of publications in the fields of music theory, musicology, and information retrieval. Before this study, the vast majority of these components had never been used in any MIR research whatsoever. The characteristics may be divided into the seven categories that are detailed below:

1. Pitch Statistics: How prevalent are certain pitches in terms of absolute pitches and pitch classes in comparison to one another? What is the piece's tonal range? How broad is the applicability of it? Is there a lot of pitch variation? What kinds of melodic intervals can you find?

2. Horizontal Intervals and Melodies: Is there a wide selection of melodies? What can be discovered by measuring the contours of melodies? What terminology is used, and how often does it appear? What kind of vertical intervals are there?

3. Vertical Intervals and Chords: What kind of chords are they meant to be? What is the magnitude and speed of the harmonic movement?

4. Rhythm: The durations of individual notes and the intervals between note attacks are used to identify characteristics. What is the rhythmic structure, and what is the metre? What are the variations in rhythm between the voices?

5. Instrumentation: What instruments are available, and which ones stand out more than others? Instruments with and without pitch are examined.

6. Texture: Is it polyphonic or homophonic? How many separate voices are present? What are the proportional advantages of different voices?

7. Dynamics: What kind of dynamics are there and how loud are the notes?

Instead of extracting all the characteristics, which might confuse the model and reduce accuracy, I simply retrieved 12 audio features that are the most relevant to music composers listed below:

1. Chromatic Motion: A semitone-equivalent fraction of melodic intervals

2. Melodic fifths: Perfect fifth-based melodic interval fractions

3. Melodic octave: A fraction of melodic intervals that are octaves

4. Melodic third: Major or minor thirds that are fraction of melodic interval

5. Prevalence of most common melodic interval: The percentage of all melodic intervals that correspond to the most prevalent melodic interval

6. Prevalence of Most Common Pitch: The proportion of notes that match the most common pitch

7. Prevalence of Most Common Pitch Class: The proportion of notes that fall within the most common pitch class

8. Relative Prevalence of Most Common Melodic Intervals: The ratio of the second-most frequent melodic interval in the composition to the most common melodic interval

9. Top Pitches Relative Prevalence: It is calculated by dividing the piece's Second Most Common Pitch's Relative Frequency by the Piece's First Most Common Pitch

10. Top Pitch Classes Relative Prevalence: It is calculated by dividing the piece's Second Most Common Pitch Class Relative Frequency by the Most Common Pitch Class Relative Frequency.

11. Repeated Notes: The percentage of melodic intervals that match repeated notes

12. Stepwise Motion: The number of melodic intervals that make up a minor or major second.

The above mentioned 12 audio features are refered from Herremans *et al.* [6]. jSymbolic can get 246 different features from a MIDI file. But for music composer recognition, the research states that not all of them are useful or easy to model. The researchers didn't include multidimensional features, nominal features, features related to instrumentation, or features that depend on the key. This led them to a choice of twelve one-dimensional features that gave out normalized frequency information about intervals or pitches. They give information about melodic intervals and pitches because they are measured as normalized frequencies. Another reason to keep the feature set small is to keep the models simple to avoid overfitting [2]. Having a small number of features lets you test a model thoroughly, which can improve the quality of a classification model [15]. CSV files were used to store these characteristics.

3.3.4 N-grams

In the case of N-grams, after reducing the data from MIDI files, the initial stage in N-gram extraction is to identify probable unigrams (making the notes in each track linear). Using time or pitch as the fundamental characteristic is the simplest method, but it does not provide appropriate results. The works are playable at a variety of tempos and in any key. The essential features must be key-independent, meaning that relative note pitch is more important than absolute note pitch. It is significant since the key of a work communicates nothing about it. For example, Johann Sebastian Bach composed two separate sets of preludes and fugues, one in each of the keys of the well-tempered scale and the other set in a different key. This resulted in a normalized, uniform pitch distribution. The second crucial property of musical N-grams is that they are tempo-insensitive. The length of MIDI files is expressed directly in a format that may be converted to milliseconds, as opposed to figuratively as quarters, eighths, or half-notes. Each MIDI file representing the same piece of music sequenced by a different person (or computer software) will change

somewhat. I opted for relative duration of counting as it is more accurate than direct duration counting. A logarithmic scale is used to quantify each difference in order to take into consideration erratic tempo variations. The equation used to apply the following pair of notes to each other is as follows:

$$(P_i, T_i) = \left(p_{i+1} - p_i, \text{round}\left(\log_2\left(\frac{t_{i+1}}{t_i}\right)\right)\right)$$

where p_i is the i_{th} note pitch (in MIDI units), t_i is the i_{th} note length (in ms), and the resultant tuple is (P_i, T_i).

A simple transition from unigrams to N-grams is to acquire n consecutive unigrams as a single item. This yields three separate sorts of N-grams. I could concentrate simply on either the melody or rhythm, or we might see N-grams as a blend of the two. The N-gram format seems similar to text. Using the Pretty MIDI library, I was able to calculate the pitch and duration of each note. Pretty MIDI offers classes and utility methods for easily editing and extracting data from MIDI files.

Chapter 4

Model Implementation

I used the Fastai package to create deep learning models [9]. Fastai is a deep learning library that offers practitioners both low-level components that may be mixed and matched to construct unique strategies, as well as high-level components for quickly and easily providing state-of-the-art results in popular deep learning domains. It strives to accomplish both objectives while maintaining performance, flexibility, and usability. Fastai was created with two main design aims in mind: to be helpful and effective while also being highly hackable and adaptable. It is supported by a base of lower-level APIs that provide modular building blocks. By doing this, it is no longer necessary for a user to get familiar with the lowest level in order to modify a component of the high-level API or add certain functionality to suit their requirements. I used Fastai to build 34-layered Resnets, Alexnet, and Squeezenet.

4.1 Resnet

Convolutional neural networks are the foundation of the cutting-edge model for image classification known as Resnet. The ImageNet dataset, a large classification dataset, was used to train these Resnet models. ImageNet has about one hundred thousand photographs divided into two hundred categories. There is a good chance that photographs are comparable to the ones (i.e. Mel Spectrograms and MFCCs) I that used for pre-training on the vast ImageNet dataset, which contains a wide diversity of image classifications. This network is based on Kaiming He *et al.* [5] paper. According to this study, adding shortcut connections to the basic network causes it to become its residual variation. The identity shortcuts may be used immediately when the input and output dimensions match. There are two options here as the dimensions increase:

1. Identity mapping is still carried out by the shortcut, but with extra zero entries to take into account the larger dimensions. No additional parameters are added by this option.

2. To ensure proportions are accurate, the projection shortcut is used (which is accomplished using 11 convolutions). Projection shortcuts take a 2 stride to cross feature maps with two sizes in both options.

Apart from the 34-layered Resnet I also implemented a Convolutional Neural Network (CNN), Squeezenet and Alexnet.

4.2 Alexnet

The paper by Krizhevsky *et al.* [13] explains Alexnet in detail. For image classification tasks, Alexnet is superior to conventional Convolutional Neural Network (CNN). The standard model for object recognition has long been convolutional neural networks (CNNs). The primary issue is that it might be difficult to make use of them on high-resolution images. At the ImageNet size, it was required to implement a GPU-optimized innovation that decreased training periods while also enhancing performance. There are a total of eight layers inside the design, including three fully linked layers and five convolutional layers. However, it is not what distinguishes AlexNet from other convolutional neural network systems. Some of its new features are as follows:

1. Nonlinearity in ReLU: AlexNet uses rectified linear units (ReLU) rather than the then-common tanh function [17]. The ReLU, or rectified linear activation function, is a linear function that gives out the input directly if it is positive and zero if it is negative. ReLU has an advantage over tanh in terms of training time; using the CIFAR-10 dataset, a CNN utilising ReLU was able to achieve a 25% error six times quicker.

2. Multiple GPUs are needed: In the past, graphics processing units (GPUs) had 3 gigabytes of memory. This was sufficient because of the fact that the training set included 1.2 million photos. In order to facilitate training on multiple GPUs, Alexnet distributes the model's neurons such that half are hosted on one GPU and the other half are hosted on another GPU. This not only makes it possible to train a bigger model, but it also speeds up the training process.

3. Overlapping Pooling: The phrase "overlapping pooling" refers to a scenario in which two or more CNNs, the outputs of adjacent clusters of neurons are

often "pooled." However, the researchers saw a 0.5% drop in error when they incorporated overlap, and they discovered that models with overlapping pooling are harder to overfit than models without overlap.

In terms of overfitting, AlexNet's 60 million parameter count posed a serious challenge. Two methods were employed to reduce overfitting:

1. Data augmentation: In order to broaden the scope of their research, the authors applied label-preserving alteration to their data. They brought about visual translations as well as horizontal reflections, both of which contributed to an increase of 2048 times in the training set. In addition to this, they altered the brightness of the RGB channels by using RGB pixel values and Principle Component Analysis (PCA), which resulted in a reduction of the top-1 mistake rate of around 1%.

2. Dropout: With the use of this method, neurons may be "turned off" with a specified rate of success (e.g., 50%). This would imply that each cycle utilizes a fresh sample of the parameters in the model, which would encourage each neuron to gain more resilient characteristics that can be utilized with other random neurons. On the other hand, dropout makes it take longer for the model to reach convergence, which is a bad thing.

The advanced model AlexNet can provide results with high accuracy on even the most challenging datasets. Any convolutional layer removal will result in a considerable decrease in AlexNet's performance. In the realm of computer vision and artificial intelligence, AlexNet is a top architecture for any task involving object detection. In the future, AlexNet may replace CNN in the picture sector.

4.3 Squeezenet

Squeezenet is a further improvement to Alexnet. The paper by Iando *et al.* [10] explains Squeezenet in detail. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50 times less parameters. Additionally, employing model compression techniques, the researchers were able to reduce SqueezeNet to less than 0.5MB. Smaller and faster model means better results with least amount of computation power. Convolutional Neural Network (CNN) research has focused mostly on enhancing accuracy

on computer vision datasets. Comparing a CNN design with more parameters to one with fewer parameters reveals a number of advantages:

1. The communication that takes place between servers in distributed CNN training limits its capacity to scale effectively. The communication that must take place during distributed data-parallel training rises in a manner that is directly proportional to the model's total number of parameters. In general, smaller models may be trained faster than larger ones because there is less need to talk to them during the training process.

2. When it comes to exporting new models to customers, there is a reduction in overhead costs. Modifications performed over-the-air to existing CNN or DNN models that are conventional may involve large data transfers. Because smaller models need fewer connections than larger ones, they may get updates more often.

3. Both embedded and Field Programmable Gate Arrays (FPGA) deployment are possible: FPGAs typically have no off-chip memory or storage and on-chip memory of less than 10MB. A suitably small model might be stored directly on the FPGA while video frames are streaming through it in real time, bypassing the memory bandwidth constraint for inference. Additionally, when CNNs are utilized on Application-Specific Integrated Circuits (ASICs), a model that is compact enough may be retained directly on-chip, which allows the ASIC to be built on a smaller die. This makes it possible for more functionality to be packed into the ASIC.

4.4 Long Short-Term Memory

Other than the above-mentioned networks, I also tried Long Short-Term Memory (LSTM). The paper by Hochreiter *et al.* [7] explains LSTM in detail. Long short-term memory (LSTM), which is similar to an artificial neural network, is used in both deep learning and artificial intelligence. LSTM neural networks have feedback connections, in contrast to the more common feedforward neural networks. In addition to processing individual data points (like photos), this kind of recurrent neural network is also

capable of managing an entire data stream (such as speech or video). For instance, LSTM might be used to enhance tasks such as unsegmented and linked handwriting identification, voice recognition, machine translation, robot control, video game design, and healthcare. The long-short-term (LSTM) neural network is the one that has been studied and developed the most during the 20th century. A typical LSTM unit as a whole is made up of the individual components that make up the cell; the input gate, the output gate, and the forget gate. The cell is able to remember values for an undetermined period of time, and the three gates that it has are responsible for managing the flow of information into and out of the cell. LSTM networks are ideal for categorizing, analyzing, and developing predictions based on time series data because there may be delays of uncertain length between significant events in a time series. This makes LSTM networks particularly well-suited for the task. This is due to the characteristics of the data being used. LSTMs were developed in order to circumvent the issue of vanishing gradients, which may arise during the training of conventional RNNs. LSTM routinely outperforms RNNs, hidden Markov models, and other sequence learning algorithms in terms of overall performance. This may be largely explained by the fact that it has a low sensitivity to the gap length.

4.5 Naïve Bayes

In the field of statistics, Naïve Bayes classifiers are a collection of uncomplicated "probabilistic classifiers" that are based on Bayes theorem and strong (naïve) independence assumptions across features. These classifiers are often referred to as simple Bayes classifiers. When combined with kernel density estimation, these Bayesian network models, which are among the simplest of all Bayesian network models, have the potential to achieve high levels of accuracy [4]. Since the method of maximum likelihood is used to estimate parameters for Naïve Bayes models, it is possible to use the Naïve Bayes model in many real-world scenarios without adopting Bayesian probability or utilizing any Bayesian procedures. The number of variables involved in a learning task, also known as features or predictors, is proportional to the number of parameters that Naïve Bayes classifiers make use of. In contrast to many other kinds of classifiers, maximum-likelihood classifiers may be taught in a linear period of time rather than via expensive iterative approximations. This makes the training

process far more efficient. Using the Naïve Bayes method, which employs models that assign problem circumstances, which are represented as vectors of feature values, a particular class label from a limited pool, it is possible to generate simple classifiers. This method is used to create classifiers. Given that all Naïve Bayes classifiers work on the premise that the value of one feature is independent of the value of any other feature, there is no single approach that can be used to train all of these classifiers. This is because, given the class variable, there is no method that can be used to train all of these classifiers. Instead, there is a set of strategies that all operate from the same fundamental assumption. An apple is a kind of fruit that is typically red in color and has a diameter of around 10 cm. A Naïve Bayes classifier holds the belief that each of the three characteristics—color, roundness, and diameter—contributes independently to the likelihood that the fruit is an apple. This is the case regardless of any possible connections that may exist between the three pieces of data. This is possible due to the fact that the Naïve Bayes model uses maximum likelihood as its estimation method. The Naïve Bayes algorithm only needs a small amount of the training data to figure out the classification parameters, which is a big plus.

Mel-Spectrograms and Mel-frequency Cepstral Coefficients (MFCC), which were saved as pictures for training, were used to train the Resnets, Squeezenet, Alexnet, CNN, and LSTM. Twelve audio characteristics that were retrieved using the jSymbolic program were trained using Naïve Bayes.

4.6 N-grams

The N-grams created from MIDI files were subjected to the N-grams method. A table including N-grams and their occurrences in each individual composer's works from the training corpus served as the basis for their profile. The computer creates a profile for the testing piece after counting every occurrence of each N-gram when a new piece is introduced to the system. Then, the profile is contrasted with the profiles of composers, and the most similar profile is picked. The intricate details of how the profiles are made and other facets of the algorithm will be covered in the following sections.

In the previous section, I explained how the N-grams are generated using the pitch and duration of notes. Fig 4.1 explains how N-grams are created from scratch.

Three different kinds of N-grams may be produced since each uni-gram consists of two values that denote changes in pitch and rhythm:

1. Melodic — when only pitch information is considered.
2. Rhythmic — if only the information about the rhythm is considered
3. Mixed — characteristics are created by both melodic and rhythmic aspects.

Relative Pitch (Melodic): 4,3,5,-1,-2,-3,-4
Relative Notes (Rhythmic):-1,0,1.6,-1.6,1,0,1

frequency

N-grams (N=3):

Melodic	Rhythmic	Combined
(4,3,5) -1	(-1,0,1.6) -1	(4,3,5,-1,0,1.6) -1
(3,5,-1) -1	(0,1.6,-1.6) -1	(3,5,-1,0,1.6,-1.6) -1
(5,-1,-2) -1	(1.6,-1.6,1) -1	(5,-1,-2,1.6,-1.6,1) -1
(-1,-2,-3) -1	(-1.6,1,0) -1	(-1,-2,-3,-1.6,1,0) -1
(-2,-3,-4) -1	(1,0,1) -1	(-2,-3,-4,1,0,1) -1

Figure 4.1: Creating profile [20]

Three different kinds of profiles may thus be formed from these N-gram categories. Certain N-grams are more frequent than others in whole chunks. N-grams serve as the keys, and occurrence counts serve as the values in each profile's table. As a result, three separate profiles are produced, and each is then independently analyzed in the stages that follow. The program then creates profiles for the files whose composer needs to be identified. Each piece is represented as three vectors of N-gram occurrences of each type in the same manner as the original profiles for the recognized piece. These vectors are compared to the appropriate composer profiles using the following similarity metric which is based on the work done by Wolkowicz *et al.* [20]:

$$Sim(\vec{x}, \vec{y}) = \sum \left(4 - \left(\frac{2 \cdot (x_i - y_i)}{x_i + y_i} \right)^2 \right)$$

where $\vec{x}$ and $\vec{y}$ denote a composer's (any type) profile and a piece's related profile.

These calculations provide $3n$ similarity values, where n is the number of composers that were analyzed and 3 is the number of profile types. The following steps were taken:

1. Add up all the similarities between each composer's profile.
2. Sort the sums in ascending order
3. Choose the composer with the highest sum as result

4.7 Architectures

4.7.1 Resnet

From Figure 4.2, the architecture starts with a convolution layer with 64 filters and 7×7 kernel followed by a max-pooling layer. After this, we have two layers of convolution layer with 64 filters and 3×3 kernel which is repeated three times. Followed by a 3×3 kernel convolution layer max-pooled with a stride of 2. Now we have two layers of convolution layers with 128 filters and 3×3 kernel, which are repeated four times. Followed by a 3×3 kernel and filter size 256 convolution layer max-pooled with a stride of 2. Followed by two layers of convolution with 256 filters and a 3×3 kernel repeated 6 times. Followed by a convolution layer with a 512 filter, max-pooled by a stride 2. Followed by two layers of convolution with 512 filters and a 3×3 kernel repeated three times. We have average pooling and softmax in the end.

4.7.2 Alexnet

From the Figure 4.3, the architecture starts with a convolution layer with 96 filters of size 11×11 with stride 4. Followed by it is a ReLU function. Then we have the Max-pooling layer, of size 3×3 and stride 2. Then in the second layer of convolution we have 256 filters with a size of 5×5 and stride is 2 and padding is 2. Relu activation function is used. Then again, we have a max pooling layer of size 3×3 with stride 2. The third convolution layer consists of 384 filters of size 3×3, stride 1, and padding 1. After that, we have ReLU activation function again. The third layer is repeated again as the fourth layer. Finally, we have a convolution layer composed of 384 filters of size 3×3, stride 1, and padding 1. followed by ReLU and max-pooling of size 3×3 and stride 2. In the end, we have a drop layer with a drop rate of 0.5. Then we have ReLU again followed by a drop layer with a rate of 0.5. Followed by ReLU and softmax activation functions in the end.

4.7.3 Squeezenet

From the Figure 4.4, we can see Squeezenet reduces the size of the filter to $1 \times 1$and the number of input channels. It begins with a convolution layer followed by eight fire modules. The Fire module consists of a convolution layer with a 1×1 filter. It is followed by ReLU. After ReLU, we have a convolution layer with a combination of 1×1 and 3×3 filters followed by ReLU. In between fire modules, we have max-pooling by a stride of 2. In the end, we have a convolution layer followed by average pooling with softmax in the end.

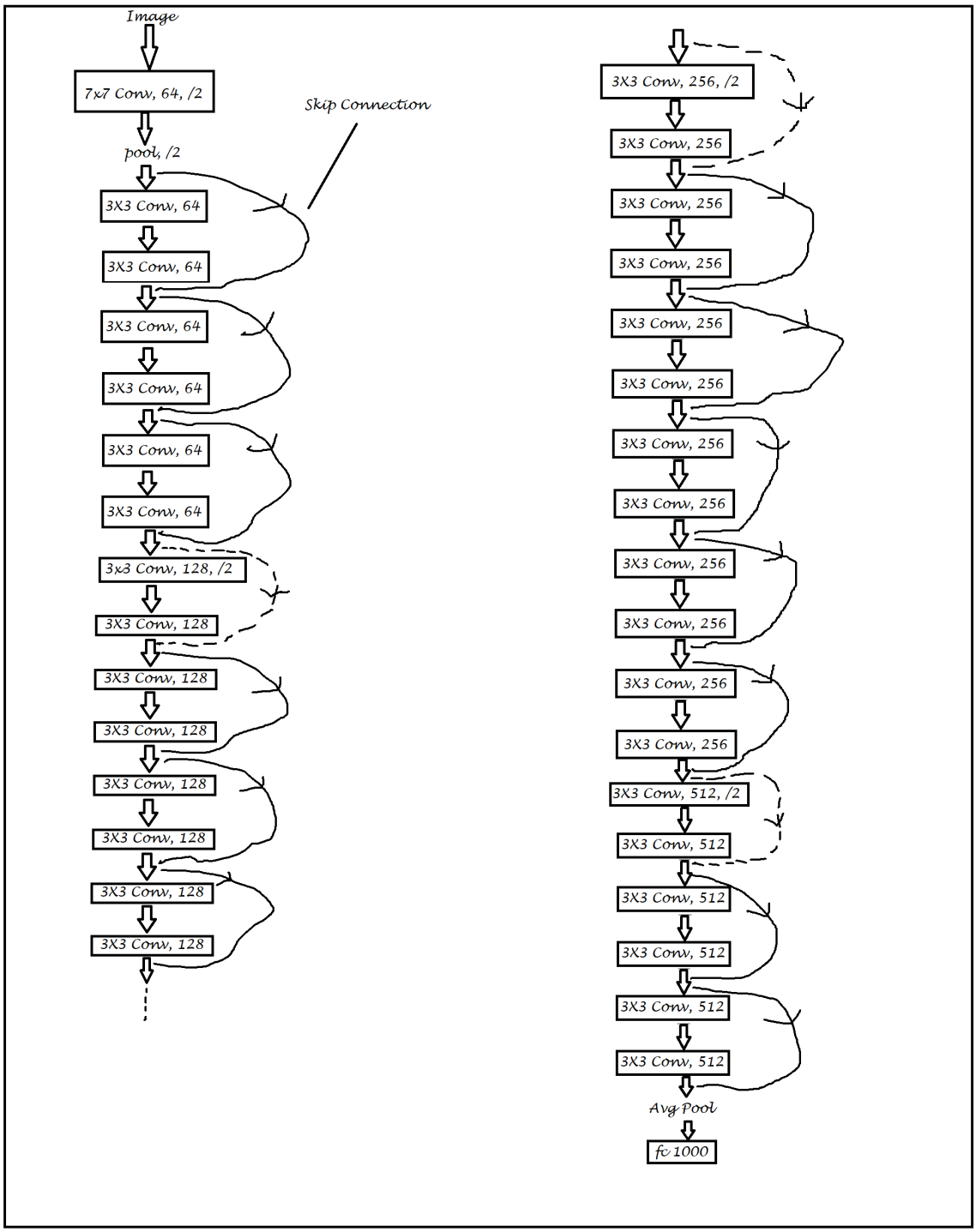

Figure 4.2: Resnet 34 [5]

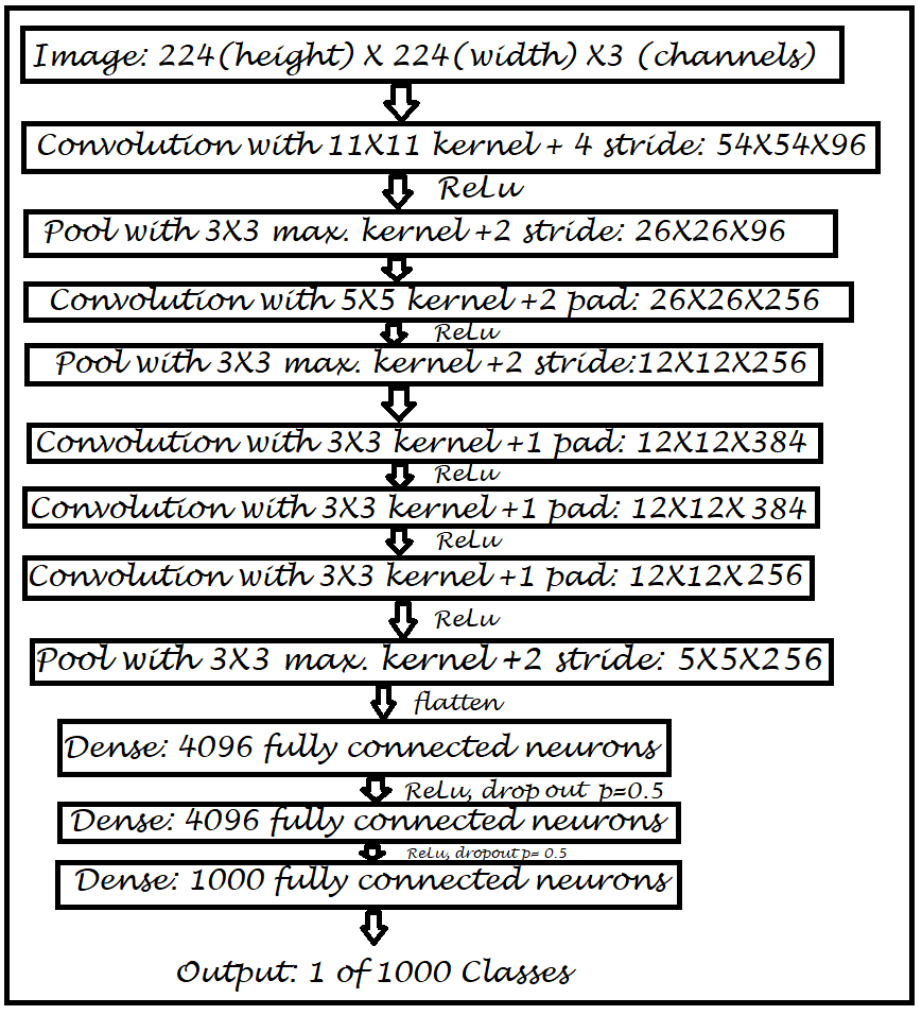

Figure 4.3: Alexnet [13]

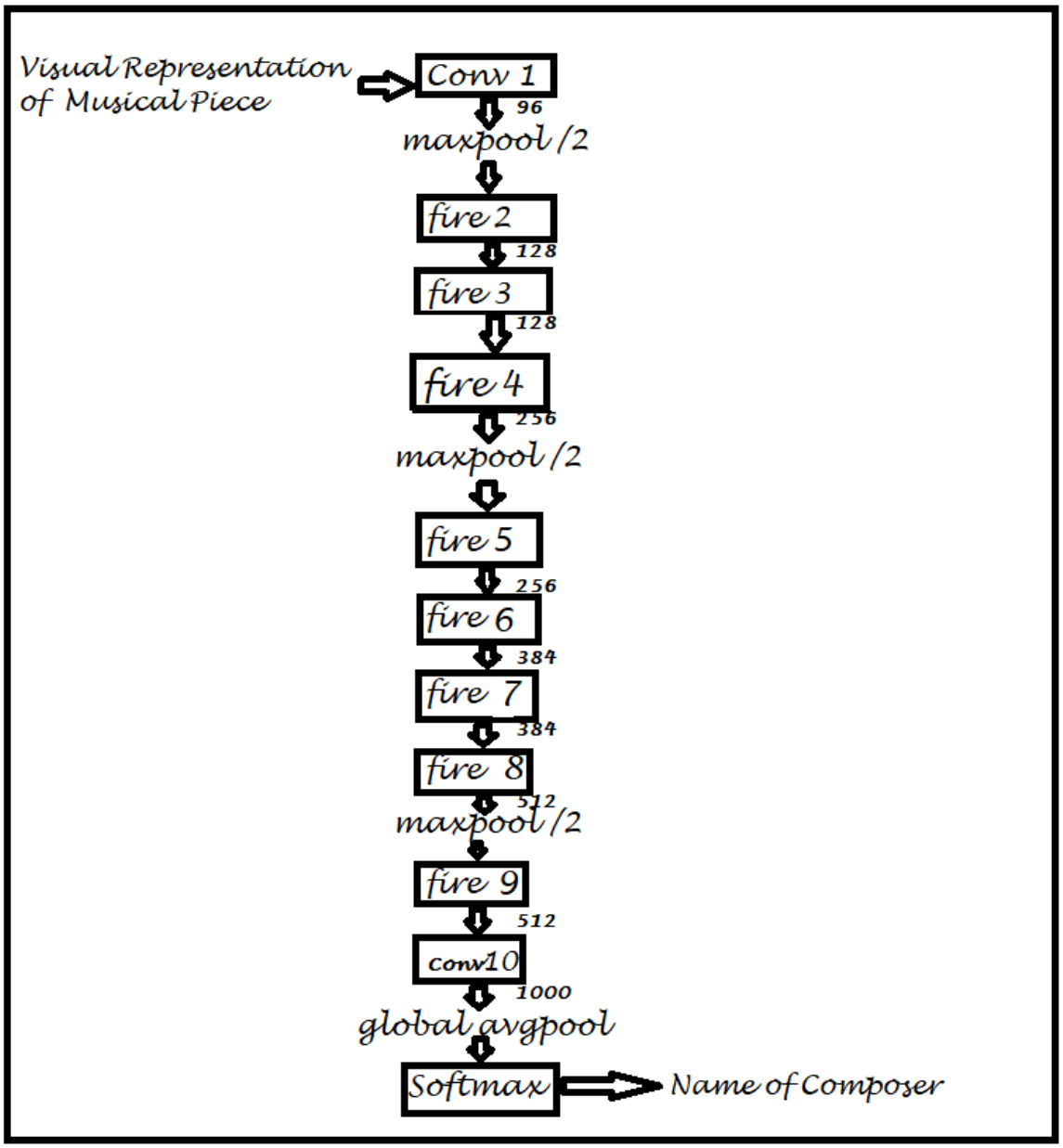

Figure 4.4: Squeezenet [10]

Chapter 5

Results and Discussion

5.1 Results

The Mel-Spectrograms and Mel-frequency Cepstral Coefficients (MFCC) were kept as images arranged in folders according to composer. With its attribute "from_folder", the Fastai function "ImageDataBunch" was employed. Fastai has a great class for handling all of the input images for vision tasks. Its name is ImageDataBunch, and it provides a range of features that match the many ways that data may be sent to the network. Images are kept in folders with names that match the image labels, hence ImageDataBunch.from_folder() method to construct an object that includes image data can be employed. This makes reading the data into our model a very simple process, which is a huge convenience. I just need to worry about three hyperparameters: the path-variable that points to our data collection, the size of the inputs, and the batch size for each iteration of the gradient descent algorithm. If any extra instructions are not specified, the ImageDataBunch object will automatically scale all pictures to a size-by-size square image. This is done to keep things as easy as possible. I'll save the ImageDataBunch object in the data variable. This object is created using the from_folder function, which was previously stated. It also needs the path to our data, the image, and the batch size. It also needs the valid pct option, which controls how many images in the validation set are chosen at random.

From its vision module, Fastai provides a function that is referred to as CNN function. This function will result in the production of a learner object, which will then be saved in a variable. Following that, the ResNet architecture will serve as the fundamental model for our transfer learning efforts. The downloaded and saved version of the trained architecture will be made available locally via the Fastai API as soon as the request is made. I'll be using the pretrained architectures of Alexnet and Squeezenet for the classification process. I chose pretrained models of these three networks because they have already been tested on 100,000 images for 200 different

classes. However, I still train them again from scratch as most of this pre-training might not be useful, i.e., all of these images on which these models are trained might not be related to our visual representations in any way. But I still chose pre-trained models because I thought that starting with pre-trained model weights might be better than starting with model weights that were picked at random. In order to validate these deep learning models, a wide range of musical composers will be used. I tested it on 11 composers of music, including Albeniz, Beethoven, Chopin, Grieg, Haydn, Liszt, Mendelssohn, Mozart, Schubert, Schumann, and Tchaikovsky. Then on to five composers, which are Beethoven, Chopin, Mozart, Schubert, and Schumann. And lastly, three composers of music: Beethoven, Chopin, and Schubert. In the cases of 11 and 5 composers, each composer was trained on 10 files, whereas in the case of 3 composers, each composer was trained on 30 files. Mel-spectrograms and Mel-frequency Cepstral Coefficients (MFCC) were used to train these models. In addition to the Deep Learning architectures listed earlier, I used machine learning models like Convolution Neural Networks (CNN) and Long Short Term Memory (LSTM) on these features. Results based on Mel-spectrogram are shown in the table 5.1, while results based on Mel-frequency Cepstral Coefficients are shown in the table 5.2.

Deep Learning Model	3 labels	5 labels	11 labels
Resnet 34	94%	80%	66%
Alexnet	76%	85%	69%
Squeezenet	94%	85%	78%
CNN	93%	65%	49%
LSTM	53%	33%	22%

Table 5.1: Results based on Mel-Spectrogram

Deep Learning Model	3 labels	5 labels	11 labels
Resnet 34	61%	52%	45%
Alexnet	85%	66%	55%
Squeezenet	61%	52%	45%
CNN	58%	45%	14%
LSTM	29%	27%	11%

Table 5.2: Results based on Mel-frequency Cepstral Coefficients (MFCC)

The N-grams algorithm was also implemented for 11, 5, and 3 composers. The

N-grams algorithm was run on the composer profiles. Here profiles were created for $N = 2$ up to $N = 12$. The algorithm was implemented for various N-gram lengths (n) and profile sizes during the composer's profile construction. The algorithm was run on melodic, rhythmic, and a combination of both for all values of N. The sum of measures of similarity of melodic, rhythmic, and combinations of both were taken into account while performing composer recognition. The table 5.3 below show the results for 11, 5, and 3 composers, respectively.

Algorithm	3 labels	5 labels	11 labels
N-grams	91%	70%	40%

Table 5.3: Results based on N-grams

The 12 audio features extracted using jSymbolic software were used for the Naïve Bayes algorithm for 3, 5, and 11 composers. Table 5.4 shows the results.

Algorithm	3 labels	5 labels	11 labels
Naïve Bayes	44%	50%	30%

Table 5.4: Results for Naïve Bayes

5.2 Discussion

5.2.1 11 Composers

Mel-Spectrograms

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for 11 composers. For 11 composers I got a accuracy of 66% for Resnet, 69% for Alexnet, 78% for Squeezenet,50% for CNN and 22% for LSTM highlighted in the table 5.5 below.

LSTM has the lowest accuracy, which is predictable as it is not suitable for pixel data. Shallow CNN, which is generally used for image data, performs decently. An improved version of CNN, namely Alexnet, designed specifically for image data, performs much better than CNN. The 34-layered Resnet, which is very deep, performs nearly as well as the Alexnet, but it requires much more computation power. Resnet

Model	Accuracy
Resnet 34	66%
Alexnet	69%
Squeezenet	78%
CNN	49%
LSTM	22%

Table 5.5: Results based on Mel-Spectrogram for 11 composers

is not suitable for this type of problem, which might be the reason for its comparatively low accuracy with respect to high expectations. Squeezenet, being a much more recent and modern neural network and a further improved version of Alexnet, has performed the best among all the neural networks with the least computation power and parameters. This also depends on a bunch of other factors, like the type of composer and the type of feature incorporated. Figures 5.1, 5.2, and 5.3 show the confusion matrix for Resnet, Alexnet, and Squeezenet, respectively, in order to understand the results in more detail.

From table 5.6, it can be seen that Resnet has performed the best for Albeniz, Mendelssohn, and Schubert and the worst for Beethoven. Accuracy for some of the composers like Tchaikovsky, Schumann, Mozart, and Haydn is quite high. The model is most confused between Mendelssohn and Liszt and between Beethoven and Schubert.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	2	1	0	0	0	0	0	0	0	0	0
Beethoven	0	1	0	0	0	0	1	0	2	0	0
Chopin	0	0	0	0	0	0	0	0	0	0	0
Grieg	0	0	1	0	0	0	0	0	1	0	0
Haydn	0	0	0	0	1	0	0	0	0	0	0
Liszt	0	0	0	0	1	0	0	0	0	0	0
Mendelssohn	0	0	0	1	0	2	3	0	0	0	1
Mozart	0	0	0	0	0	0	0	1	0	0	0
Schubert	0	0	0	0	0	0	0	0	2	0	0
Schumann	0	0	0	0	0	0	0	0	0	1	0
Tchaikovsky	0	0	0	0	0	0	0	0	0	0	1

Table 5.6: Confusion matrix for Resnet on Mel-Spectrogram for 11 composers

From table 5.7, it can be seen that Alexnet has performed the best for Albeniz, Mendelssohn and the worst for Beethoven. Accuracy for some of the composers like Schubert, Mozart, and Haydn is quite high. The model is most confused between Mendelssohn and Haydn and between Beethoven and Schubert.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	3	0	0	0	0	0	0	0	0	0	0
Beethoven	0	1	0	0	0	0	0	0	2	0	0
Chopin	0	0	0	0	0	0	0	0	0	0	0
Grieg	0	0	0	0	0	0	0	0	1	0	0
Haydn	0	0	0	0	2	0	0	0	0	0	0
Liszt	0	0	0	0	0	1	0	0	0	0	0
Mendelssohn	0	0	0	0	1	0	5	0	0	0	1
Mozart	0	0	0	0	0	0	0	2	0	0	0
Schubert	0	0	0	0	0	0	0	0	2	0	0
Schumann	0	0	0	0	0	1	0	0	0	1	0
Tchaikovsky	0	0	0	0	0	1	0	0	0	0	0

Table 5.7: Confusion matrix for Alexnet on Mel-Spectrograms for 11 composers

From table 5.8, it can be seen that Squeezenet has performed the best for Mendelssohn and the worst for Tchaikovsky and Schumann. Accuracy for some of the composers like Beethoven, Albeniz, Mozart, and Schubert is quite high. The model is most confused between Tchaikovsky and Albeniz and between Beethoven and Schumann.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	2	0	0	0	0	0	0	0	0	0	1
Beethoven	0	2	0	0	0	0	0	0	0	1	0
Chopin	0	0	0	0	0	0	0	0	0	0	0
Grieg	0	0	0	0	0	0	0	0	0	0	0
Haydn	0	0	0	0	1	0	0	0	1	0	0
Liszt	0	0	0	0	0	1	0	0	0	0	0
Mendelssohn	0	0	0	0	0	0	6	0	0	1	0
Mozart	0	0	0	0	0	0	0	2	0	0	0
Schubert	0	0	0	0	0	0	0	0	2	0	0
Schumann	0	0	0	0	0	1	0	1	0	1	0
Tchaikovsky	1	0	0	0	0	1	0	0	0	0	0

Table 5.8: Confusion matrix for Squeezenet on Mel-Spectrograms for 11 composers

One thing is pretty clear from all the above results: Mendelssohn is the most easily recognized composer with the highest accuracy as far as Mel-Spectrograms are concerned. Beethoven is among the most difficult to classify. The very high accuracy of Mendelssohn is primarily responsible for the overall high accuracy.

12 audio features extracted using jSymbolic

I also implemented Naïve Bayes using the 12 audio features extracted using jSymbolic software. The table 5.9 below shows the accuracy for 11 composers. Naïve Bayes had

Model	Accuracy
Naïve Bayes	30%

Table 5.9: Results for Naïve Bayes on 11 composers

an accuracy of 30% for 11 composers.

Mel-frequency Cepstral Coefficients

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for 11 composers. I got a accuracy of 45% for Resnet, 55% for Alexnet, 45% for Squeezenet, 14% for CNN, and 11% for LSTM highlighted in the table 5.10 below.

Model	Accuracy
Resnet 34	45%
Alexnet	55%
Squeezenet	45%
CNN	14%
LSTM	11%

Table 5.10: Results on Mel-frequency Cepstral Coefficients for 11 composers

The tables 5.11, 5.12 and 5.13 show the confusion matrix for Resnet, Alexnet, and Squeezenet, respectively, in order to understand the results in more detail. Among the three, Resnet, Squeezenet are at the same level, while Alexnet performs slightly better than the rest. CNN and LSTM have performed quite poorly. The accuracy of CNN and LSTM is slightly higher than the random probability which is 8.3%.

From the table 5.11, it can be seen that Resnet has performed the best for Chopin and Mozart and the worst for Mendelssohn, Albeniz, Haydn, and Schumann. Accuracy for some of the composers, like Grieg, is quite high. The model is most confused between Mendelssohn and Grieg.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	0	0	0	1	1	0	0	0	0	0	1
Beethoven	0	0	0	0	0	0	0	0	0	0	0
Chopin	0	0	2	0	0	0	0	0	0	0	0
Grieg	0	0	0	1	0	0	0	0	0	0	0
Haydn	0	0	1	0	1	0	0	1	0	0	0
Liszt	0	0	0	0	0	0	0	0	0	0	0
Mendelssohn	1	0	0	2	0	0	0	0	1	0	0
Mozart	0	0	0	0	0	0	0	2	0	0	0
Schubert	0	0	0	0	0	0	0	0	0	0	0
Schumann	0	0	1	1	0	0	1	0	1	0	0
Tchaikovsky	0	1	1	0	0	0	0	0	0	0	1

Table 5.11: Confusion matrix for Resnet on Mel-frequency Cepstral Coefficients for 11 composers

From the tables 5.12, it can be seen that Alexnet has performed the best for Grieg and Mozart and the worst for Mendelssohn, Tchaikovsky, and Schumann. The model is most confused between Mendelssohn and Albeniz, and between Schumann and Chopin, and between Tchaikovsky and Beethoven.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	0	1	0	0	0	0	1	0	0	0	1
Beethoven	0	0	0	0	0	0	0	0	0	0	0
Chopin	1	0	1	0	0	0	0	0	0	0	0
Grieg	0	0	0	1	0	0	0	0	0	0	0
Haydn	0	1	1	0	1	0	0	0	0	0	0
Liszt	0	0	0	0	0	0	0	0	0	0	0
Mendelssohn	2	1	0	0	0	0	0	0	0	1	0
Mozart	0	0	0	0	0	0	0	1	0	1	0
Schubert	0	0	0	0	0	0	0	0	0	0	0
Schumann	0	0	2	0	0	1	0	0	0	0	1
Tchaikovsky	0	1	0	0	0	0	0	0	0	1	0

Table 5.12: Confusion matrix for Alexnet on Mel-frequency Cepstral Coefficients for 11 composers

From the tables 5.13, it can be seen that Squeezenet has performed the best for Haydn and the worst for Tchaikovsky and Albeniz. Accuracy for some of the composers, like Chopin, is quite high. The model is most confused between Tchaikovsky and Beethoven.

	Predicted										
	Albeniz	Beethoven	Chopin	Grieg	Haydn	Liszt	Mendelssohn	Mozart	Schubert	Schumann	Tchaikovsky
Albeniz	0	0	0	1	0	0	1	0	0	1	1
Beethoven	0	0	0	0	0	0	0	0	0	0	0
Chopin	0	0	1	0	0	0	0	0	0	1	0
Grieg	0	0	0	1	0	0	0	0	0	0	0
Haydn	0	0	1	0	2	0	0	0	0	0	0
Liszt	0	0	0	0	0	0	0	0	0	0	0
Mendelssohn	1	1	0	1	0	0	1	0	0	0	0
Mozart	1	0	0	0	0	0	0	1	0	0	0
Schubert	0	0	0	0	0	0	0	0	0	0	0
Schumann	0	0	1	0	0	1	0	0	1	1	1
Tchaikovsky	0	2	0	0	0	0	0	0	0	1	0

Table 5.13: Confusion matrix for Squeezenet on Mel-frequency Cepstral Coefficients for 11 composers

N-grams

I also implemented N-grams for 11 composers whose profiles were created using the method that was stated in the earlier sections. The N-grams algorithm was implemented for values of N ranging from 2 to 3. In the table 5.14 below, it can be seen that the accuracy decreases as the value of N increases.

N	Correct	Wrong	Accuracy
2	4	6	40%
3	3	7	30%
4	4	6	40%
5	3	7	30%
6	3	7	30%
7	3	7	30%
8	2	8	20%
9	2	8	20%
10	2	8	20%
11	2	8	20%
12	2	8	20%

Table 5.14: Results on 11 composers for complete profile length

The table 5.14 was implemented on the complete length of composer profiles. The N-grams in the profile are arranged in the order of their frequency, with the most repeated topping the list and so on. There is a possibility that the least frequent N-grams can confuse the model, leading to lower accuracy. Therefore, I decided to implement the N-grams algorithms on the first 100, 250, 500, 1000, and 2000 most frequent N-grams. It is important to note that the average length of a composer's profile is around 10,000. So basically, I'm taking into account only the top 20% of the N-grams. For 11 composer the high accuracy was 40% for complete profile length and for $N = 2$. But in the case of most frequent, the accuracy was 50% for top 2000 and 1000 N-grams for $N = 2$ and same for the top 500 N-grams for $N = 5$ which can be seen in the table 5.15 below.

5.2.2 5 Composers

Mel-Spectrograms

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for 5 composers. I got an accuracy of 80% for Resnet, 85% for Alexnet, 85% for Squeezenet, 65% for CNN and 33% for LSTM highlighted in the table 5.16 below.

Like in the case of 11 composers, Alexnet and Squeezenet have performed the best, and Resnet is not too far behind. Shallow networks like CNN have also performed

N	100	250	500	1000	2000
2	20%	40%	40%	50%	50%
3	30%	30%	30%	30%	30%
4	30%	40%	30%	40%	40%
5	40%	40%	50%	40%	30%
6	30%	20%	30%	20%	30%
7	20%	20%	30%	20%	20%
8	10%	10%	10%	10%	10%
9	10%	10%	0	10%	0
10	10%	10%	0	0	0
11	20%	10%	10%	10%	10%
12	20%	20%	10%	20%	20%

Table 5.15: Results on 11 composers for varying profile length like top 100, 250, 500, 2000 frequently repeated N-grams

Model	Accuracy
Resnet 34	80%
Alexnet	85%
Squeezenet	85%
CNN	65%
LSTM	33%

Table 5.16: Results on Mel-Spectrograms for 5 composers

decently. LSTM has performed the worst.

From the figure 5.17, it can be seen that Resnet has performed the best for Beethoven and the worst for Schubert, Mozart, and Schumann. Apart from Beethoven, Chopin also has decent accuracy. The model is most confused between Schubert and Beethoven.

	Predicted				
	Beethoven.	Chopin	Mozart	Schubert	Schumann
Beethoven.	3	0	0	1	0
Chopin	0	1	0	0	0
Mozart	1	0	0	0	0
Schubert	2	0	0	1	0
Schumann	1	0	0	0	0

Table 5.17: Confusion matrix for Resnet 34 on Mel-Spectrograms for 5 composers

From the figure 5.18, it can be seen that Alexnet has performed the best for

Schubert and the worst for Beethoven and Chopin. Apart from Schubert, Mozart also has decent accuracy. The model is most confused between Beethoven and Mozart.

		Predicted			
	Beethoven	Chopin	Mozart	Schubert	Schumann
Beethoven	2	0	2	0	0
Chopin	0	0	0	0	1
Mozart	0	0	1	0	0
Schubert	0	0	0	2	1
Schumann	0	0	0	0	1

Table 5.18: Confusion matrix for Alexnet on Mel-Spectrograms for for 5 composers

From the table below 5.19, it can be seen that Squeezenet has performed the best for Beethoven and Schubert. The model is most confused between Schubert and Beethoven.

		Predicted			
	Beethoven	Chopin	Mozart	Schubert	Schumann
Beethoven	4	0	0	0	0
Chopin	0	1	0	0	0
Mozart	0	0	1	0	0
Schubert	1	0	0	2	0
Schumann	0	0	0	0	1

Table 5.19: Confusion matrix for Squeezenet on Mel-Spectrograms for 5 composers

12 audio features extracted using jSymbolic

I also implemented Naïve Bayes using the 12 audio features extracted using jSymbolic software. The table 5.20 below shows the accuracy for 5 composers.

Model	Accuracy
Naïve Bayes	50%

Table 5.20: Results on Naïve Bayes for 5 composers

Naïve Bayes had an accuracy of 50% for 5 composers. This accuracy is quite low but still significantly higher than the random probability, which is 20%.

Mel-frequency Cepstral Coefficients

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for five composers. I got an accuracy of 52% for Resnet, 66% for Alexnet, 52% for Squeezenet, 45% for CNN and 27% for LSTM highlighted in the table 5.21 below. As seen in the case of 11 composers,

Model	Accuracy
Resnet 34	52%
Alexnet	66%
Squeezenet	52%
CNN	45%
LSTM	27%

Table 5.21: Results on Mel-frequency Cepstral Coefficients for 5 composers

Alexnet has performed better than the rest when models are trained using Mel-frequency Cepstral Coefficients (MFCC). The same is seen in the case of 5 composers. Resnet and Squeeznet are far behind. CNN is not far behind Resnet and Alexnet. Similarly, LSTM has performed the worst.

From the table 5.22 below, it can be seen that Resnet has performed the best for Chopin and Schubert and the worst for Beethoven and Schumann. The model is most confused between Schumann and Chopin, between Schubert and Mozart, and between Beethoven and Mozart.

	Predicted				
	Beethoven	Chopin	Mozart	Schubert	Schumann
Beethoven	1	1	2	0	0
Chopin	1	4	0	0	0
Mozart	1	0	1	0	0
Schubert	0	0	2	4	0
Schumann	1	3	0	0	0

Table 5.22: Confusion matrix for Resnet on Mel-frequency Cepstral Coefficients for 5 composers

From the table 5.23 below, it can be seen that Alexnet has performed the best for Chopin and Schumann and the worst for Schubert. The model is most confused between Schubert and Beethoven, between Schubert and Mozart, and between Chopin and Schumann.

	Predicted				
	Beethoven	Chopin	Mozart	Schubert	Schumann
Beethoven	2	1	1	0	0
Chopin	0	3	0	0	2
Mozart	0	0	1	0	1
Schubert	2	0	3	1	0
Schumann	0	1	0	0	3

Table 5.23: Confusion matrix for Alexnet on Mel-frequency Cepstral Coefficients for 5 composers

From the table 5.24 below, it can be seen that Squeezenet has performed the best for Chopin and Schubert and the worst for Schumann. The model is most confused between Beethoven and Mozart, and also between Schubert and Mozart, and between Schumann and Chopin.

	Predicted				
	Beethoven.	Chopin	Mozart	Schubert	Schumann
Beethoven.	1	1	2	0	0
Chopin	1	4	0	0	0
Mozart	1	0	1	0	0
Schubert	0	0	2	4	0
Schumann	1	3	0	0	0

Table 5.24: Confusion matrix for Squeezenet on Mel-frequency Cepstral Coefficients for 5 composers

N-gram

I also implemented N-grams for five composers whose profiles were created using the method that was stated in the earlier sections. The N-grams algorithm was implemented for values of N ranging from $N = 2$ to $N = 12$. In the table below, it can be seen that the accuracy decreases as the value of N initially increases, then increases again for intermediate values of N, and then again decreases. The accuracy depends on the length of N-grams and also on the composers. Certain composers have a higher suitability with a particular number of N-grams.

The table 5.25 was implemented on the complete length of composer profiles. I decided to implement the N-grams algorithms on the first 100, 250, 500, 1000, and 2000 most frequent N-grams. It is important to note that the average length of a composer's profile is around 10,000. For five composers, the high accuracy was 70%

N	Correct	Wrong	Accuracy
2	7	3	70%
3	5	5	50%
4	5	5	50%
5	5	5	50%
6	6	4	60%
7	6	4	60%
8	5	5	50%
9	5	5	50%
10	5	4	55.5%
11	4	5	44.4%
12	4	5	44.4%

Table 5.25: Results on 5 composers for complete profile length

for the complete profile length and for $N = 2$. But in the case of most frequent, the accuracy was 70% for top 2000 and 1000 N-grams for $N = 2$ and for the top 2000 N-grams for $N = 5$ which can be seen in the table 5.26 below.

N	100	250	500	1000	2000
2	60%	40%	60%	70%	70%
3	30%	50%	40%	50%	40%
4	50%	60%	50%	50%	50%
5	50%	60%	60%	50%	50%
6	50%	40%	50%	40%	70%
7	40%	50%	60%	40%	60%
8	33.3%	33.3%	44.4%	33.3%	33.3%
9	22.2%	22.2%	22.2%	33.3%	22.2%
10	22.2%	22.2%	22.2%	22.2%	22.2%
11	33.3%	33.3%	33.3%	33.3%	33.3%
12	33.3%	33.3%	33.3%	33.3%	44.4%

Table 5.26: Results on 5 composers for varying profile length like top 100, 250, 500, 2000 frequently repeated N-grams

5.2.3 3 Composers

Mel-Spectrogram

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for 3 composers. I got an accuracy

of 94% for Resnet, 76% for Alexnet, 94% for Squeezenet, 93% for CNN and 53% for LSTM highlighted in the table 5.27 below.

Model	Accuracy
Resnet 34	94%
Alexnet	76%
Squeezenet	94%
CNN	93%
LSTM	53%

Table 5.27: Results on Mel-Spectrogram for 3 composers

Like in the case of five composers, Resnet and Squeezenet have performed the best, and Alexnet is too far behind. Shallow networks like CNN have also performed at par with Squeezenet and Resnet. One of the major reasons that these neural networks have performed slightly differently than in the cases of 5 and 11 composers is that there are fewer composers to recognize, making it easier. Similar to in the cases of 11 and 5 composers, LSTM has performed the worst.

From the table 5.28 below, it can be seen that Resnet has performed the best for Schubert. The model has performed the best for all due to the lower number of composers and higher availability of data.

	Predicted		
	Beethoven	Chopin	Schubert
Beethoven	4	1	0
Chopin	0	4	1
Schubert	1	0	6

Table 5.28: Confusion matrix for Resnet on Mel-Spectrogram for 3 composers

From the table 5.29 below, it can be seen that Alexnet has performed the best for Beethoven and Chopin and the worst for Schubert. The model is most confused between Schubert and Beethoven.

From the table 5.30 below, it can be seen that Squeeznet has performed the best for Beethoven and Chopin and the worst for Schubert. The model is most confused between Schubert and Beethoven.

	Predicted		
	Beethoven	Chopin	Schubert
Beethoven	5	0	0
Chopin	0	5	0
Schubert	5	0	2

Table 5.29: Confusion matrix for Alexnet on Mel-Spectrogram for 3 composers

	Predicted		
	Beethoven	Chopin	Schubert
Beethoven	3	0	2
Chopin	0	7	2
Schubert	1	1	5

Table 5.30: Confusion matrix for Squeezenet on Mel-Spectrogram for 3 composers

12 audio features extracted using jSymbolic

I also implemented Naïve Bayes using the 12 audio features extracted using jSymbolic software. The table 5.31 below shows the accuracy for three composers. Naïve Bayes

Model	Accuracy
Naïve Bayes	44%

Table 5.31: Results of Naïve Bayes for 3 composers

had an accuracy of 44% for 3 composers, which is slightly higher than the random probability of 33%.

Mel-frequency Cepstral Coefficients

I implemented deep learning and machine learning architectures like the 34-layered Resnet, Alexnet, Squeezenet, CNN, and LSTM for 3 composers. I got an accuracy of 61% for Resnet, 85% for Alexnet, 61% for Squeezenet, 58% for CNN and 29% for LSTM highlighted in the table 5.32 below. As seen in the case of five composers, Alexnet has performed better than the rest when models are trained using Mel-frequency Cepstral Coefficients (MFCC). Resnet and Squeeznet are far away in terms of accuracy. CNN is not far behind Resnet and Squeezenet. Similarly, LSTM has performed the worst.

From the table 5.33 below, it can be seen that Resnet has performed the best for Chopin and the worst for Schubert. The model is most confused between Schubert

Model	Accuracy
Resnet 34	61%
Alexnet	85%
Squeezenet	61%
CNN	58%
LSTM	29%

Table 5.32: Results of Mel-frequency Cepstral Coefficients for 3 composers

and Beethoven and Beethoven and Schubert.

	Predicted		
	Beethoven	Chopin	Shubert
Beethoven	2	1	2
Chopin	1	8	0
Shubert	4	1	2

Table 5.33: Confusion matrix for Resnet on Mel-frequency Cepstral Coefficients for 3 composers

From the table 5.34 below, it can be seen that Alexnet has performed the best for Chopin and Beethoven and the worst for Schubert. The model is most confused between Schubert and Beethoven.

	Predicted		
	Beethoven	Chopin	Shubert
Beethoven	4	0	1
Chopin	1	8	0
Shubert	3	1	3

Table 5.34: Confusion matrix for Alexnet on Mel-frequency Cepstral Coefficients for 3 composers

From the table 5.35 below, it can be be seen that Squeezenet has performed the best for Chopin and Schubert and the worst for Beethoven. The model is most confused between Beethoven and Schubert and between Chopin and Schubert.

N-grams

I also implemented N-grams for 3 composers whose profiles were created using the method that was stated in the earlier sections. The N-grams algorithm was implemented for values of N ranging from $N = 2$ to $N = 12$. In the table 5.36 below,

	Predicted		
	Beethoven	Chopin	Shubert
Beethoven	3	0	2
Chopin	0	7	2
Shubert	1	1	5

Table 5.35: Confusion matrix for Squeezenet on Mel-frequency Cepstral Coefficients for 3 composers

it can be seen that the accuracy decreases as the value of N initially increases, then increases again for intermediate values of N, and then again decreases.

N	Correct	Wrong	Accuracy
2	8	4	66.6%
3	7	5	58.3%
4	8	4	66.66%
5	10	2	83.3%
6	11	1	91.6%
7	11	1	91.6%
8	10	2	83.3%
9	9	3	75%
10	7	5	58.3%
11	5	7	41.6%
12	5	7	41.6%

Table 5.36: Results on 3 composers for complete profile length

Table 5.36 shows the results on the complete length of composer profiles. I decided to implement the N-grams algorithms on the first 100, 250, 500, 1000, and 2000 most frequent N-grams. For 3 composers, the high accuracy was 91.6% for the complete profile length and for $N = 6$ and $N = 7$. But in the case of the most frequent, the accuracy was as high as 100% for the top 250 for $N = 5$ and 91.6% for 1000 N-grams for $N = 5$ as shown in the table 5.37 below.

The figure 5.2 below highlights the performance of all the methods for 3, 5, and 11 composers for all types of features. It can be seen that Squeezenet with Mel Spectrograms performed the best, followed by Alexnet with MFFCs. N-grams are not much behind in the cases of three composers. CNN and LSTM have performed the worst. Naïve Bayes has the overall worst performance.

N	100	250	500	1000	2000
2	50%	33.3%	75%	75%	66.6%
3	66.6%	58.3%	66.6%	75%	58.3%
4	75%	66.6%	75%	83.3%	66.6%
5	75%	100%	83.3%	91.6%	75%
6	66.6%	75%	75%	83.3%	83.3%
7	66.6%	75%	75%	75%	91.6%
8	75%	83.3%	66.6%	50%	58.3%
9	66.6%	58.3%	58.3%	50%	58.3%
10	58.3%	58.3%	58.3%	66.6%	50%
11	58.3%	58.3%	50%	66.6%	58.3%
12	50%	50%	50%	50%	50%

Table 5.37: Results on 3 composers for varying profile length like top 100, 250, 500, 2000 frequently repeated N-grams

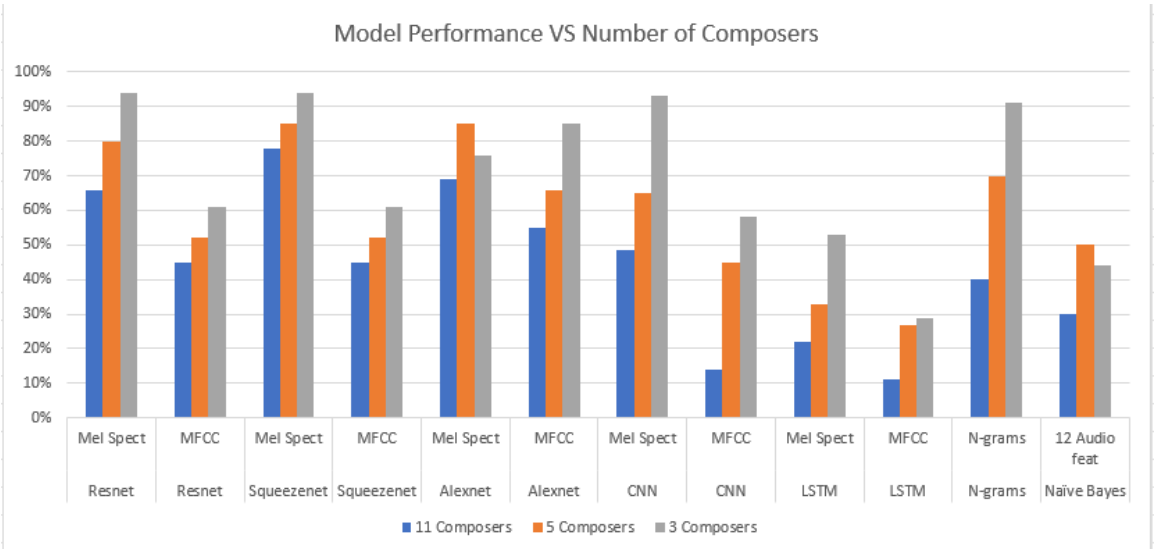

Figure 5.1: Analysis of various models against Number of Composers

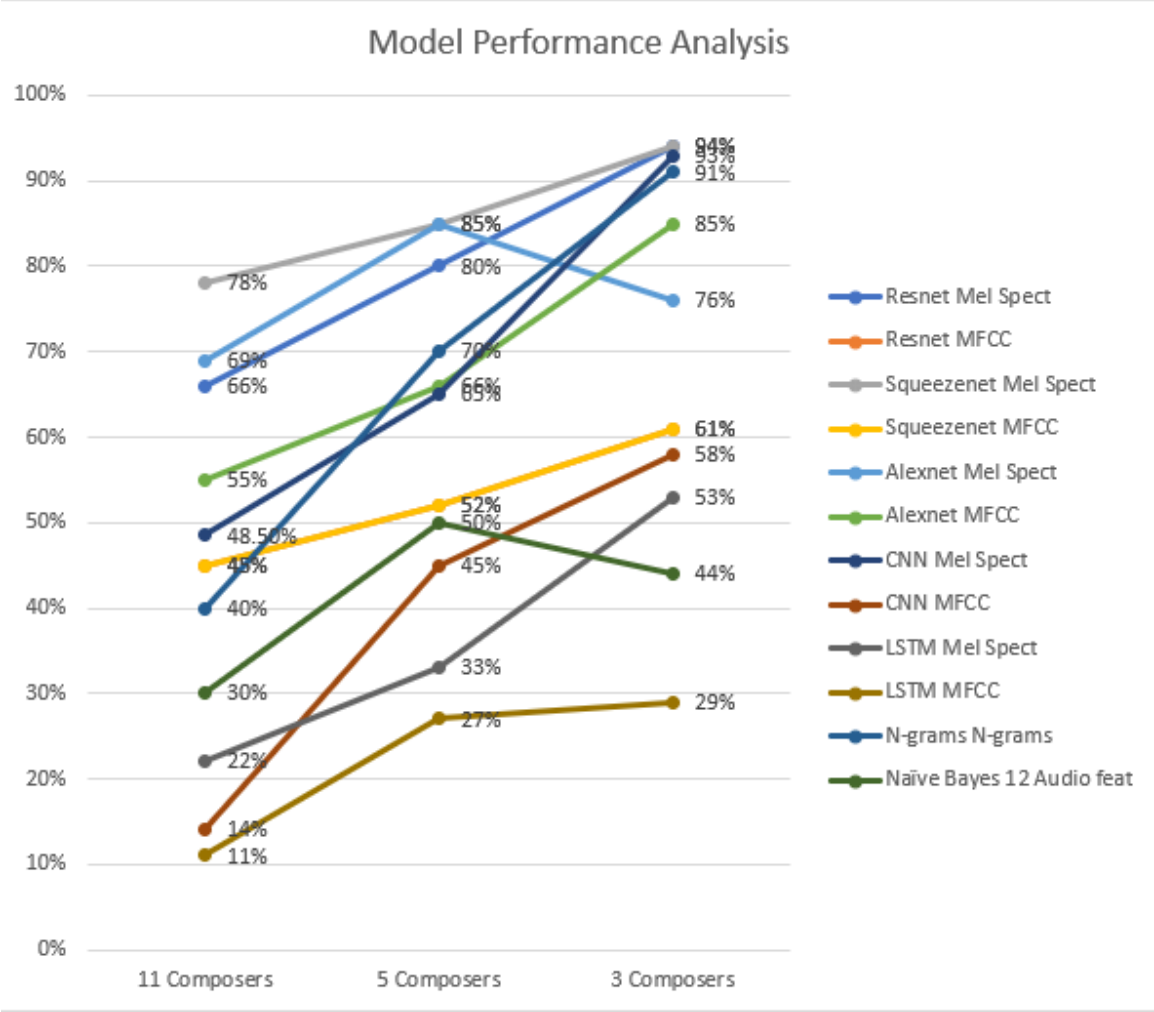

Figure 5.2: Performance Graph

In order to understand it more clearly, I created the tables 5.38 and 5.39 where I wrote down the composers with high accuracy, composers with low accuracy, and the most confused set of pairs for different deep learning models and different features. It is evident that certain models and features have high accuracy for certain composers and low accuracy for certain composers. Keeping a model constant and then looking at results across different numbers of composers and features, we can definitely see patterns with respect to the most confused pairs and the best and worst accuracy for particular composers. Similarly, when we keep a feature constant and then look at results across different numbers of composers and models, we can definitely see patterns with respect to the most confused pairs and the best and worst accuracy for particular composers. After observing the table 5.38 and 5.39 below, I can clearly see a pattern with respect to a particular model and a particular set of features. From the tabletables 5.38 and 5.39, it is pretty evident that Mel-Spectrograms have performed the best for Mendelssohn, Schubert, and Chopin, while the worst has been for Beethoven. Similarly, MFCCs have performed best for Schubert and Chopin, and their worst performances were for Mendelssohn, Schumann, and Beethoven. In the case of deep learning models, Resnet has performed best for Schubert and worst for Schumann and Beethoven. Similarly, Alexnet has performed the best for Chopin and the worst for Beethoven and Schubert. Squeezenet has performed the best with Chopin and Schubert, and Tchaikovsky is the worst one.

Model	11 Composers		5 Composers		3 Composers	
	Spect	MFCC	Spect	MFCC	Spect	MFCC
Resnet	Best:	Best:	Best:	Best:	Best:	Best:
	Albeniz Mendelssohn Schubert	Chopin Mozart	Beethoven	Chopin Schubert	Schubert	Chopin
	Worst:	Worst:	Worst:	Worst:	Worst:	Worst:
	Beethoven	Mendelsshon	Schubert Mozart Schumann	Beethoven Schumann		Beethoven Schubert
	Confusion:	Confusion:	Confusion:	Confusion	Confusion:	Confusion
	Mendelsshon ↓ Liszt Beethoven ↓ Schubert	Mendelsshon ↓ Grieg	Schubert ↓ Beethoven	Schumann ↓ Chopin Schubert ↓ Mozart	Beethoven ↓ Chopin Chopin ↓ Schubert Schubert ↓ Beethoven	Schubert ↓ Beethoven Beethoven ↓ Schubert

Table 5.38: Resnet model performance analysis

Alexnet	Best:	Best:	Best:	Best:	Best:	Best:
	Albeniz Mendelssohn	Grieg Mozart	Schubert	Chopin Chopin	Beethoven Chopin	Chopin
	Worst:	Worst:	Worst:	Worst:	Worst:	Worst:
	Beethoven	Mendelssohn Tchaikovsky Schumann	Beethoven Chopin	Schubery	Schubert	Schubert
	Confusion:	Confusion:	Confusion:	Confusion:	Confusion:	Confusion:
	Mendelssohn ↓ Haydn Beethoven ↓ Schubert	Mendelssohn ↓ Albeniz Schumann ↓ Chopin Tchaikovsky ↓ Beethoven	Beethoven ↓ Mozart	Schubert ↓ Beethoven Schubert ↓ Mozart Chopin ↓ Schumann	Schubert ↓ Beethoven	Schubert ↓ Beethoven
Squee-zenet	Best:	Best:	Best:	Best:	Best:	Best:
	Mendelssohn	Haydn	Beethoven Schubert		Chopin	Chopin Schubert
	Worst:	Worst:	Worst:	Worst:	Worst:	Worst:
	Tchaikovsky Schumann	Tchaikovsky Albeniz		Schumann	Beethoven	Beethoven
	Confusion:	Confusion:	Confusion:	Confusion:	Confusion:	Confusion:
	Tchaikovsky ↓ Albeniz	Tchaikovsky ↓ Beethoven		Beethoven ↓ Mozart Schubert ↓ Mozart Schumann ↓ Chopin	Beethoven ↓ Schubert Chopin ↓ Schubert Schubert ↓ Beethoven Schubert ↓ Chopin	Beethoven ↓ Schubert Chopin ↓ Schubert

Table 5.39: Alexnet and Squeezenet model performance analysis

Chapter 6

Conclusion

I tried to present different machine learning and deep learning techniques and different types of features for music composer recognition. Here, I also tried different types of models and feature combinations. The results of the research conducted show that there are multiple constraints or factors that affect the result, such as the type of model used, the type of feature used, the composer, and the number of composers. It is evident from the results that Mel-Spectrograms are the most suitable type of feature for composer recognition. Mel-Spectrograms along with Squeezenet and Resnet provide high accuracy. As the number of composers to be recognized goes down, it is clear that the accuracy goes up.

One of the main issues during this thesis was getting the right data. Ideally, it is important to have a sufficient amount of data in order to implement different kinds of machine learning or deep learning models. However, in this case, I had access to a satisfactory amount of data. So I tried to optimize the models in order to get the best possible results. It can be evidently seen from the results that certain models are more suitable for certain types of composers, and similarly, certain features are able to highlight or recognize specific composers distinctly. In the same way, certain models give better results with a specific set of features. So an optimal combination of models, features, and composers will give the best results.

N-grams is a Natural Language Processing (NLP) methodology which was also implemented for composer recognition. N-grams are not suitable for this kind of problem. N-grams actually performed quite well, giving 40% for 11 composers, 70% for 5 composers, and 91% for 3 composers. Here I observed that the amount of data required is directly proportional to the number of composers. As the algorithm creates N-grams of rhythm, melody, and combinations of rhythm and melody. A large number of music files will help in creating a large profile of N-grams of the composers, leading to higher accuracy. The other two factors are the value of N and the size of

the composer profile, i.e., whether all N-grams should be taken into consideration or only the most frequently repeated N-grams should be taken into consideration. It has been observed that lower and some high intermediate values of N have given the most optimal results. As far as profile length is considered, the top 1000 and 2000 most frequently repeated N-grams give the best results. For 11 composers, the accuracy is low, but for 5 and 3 composers, the accuracy is quite high. Also, the accuracy for 3 composers is on par with that of machine learning models like Squeezenet and deep learning models like Resnet. Secondly, N-grams do not require high computation power compared to the above mentioned models. For five composers, the N-grams accuracy is 70% which is slightly lower than Squeezenet, Resnet, and Alexnet.

If we look at the results of Resnet, Squeezenet, Alexnet, CNN, and LSTM on Mel-Spectrograms and Mel-frequency cepstral coefficients (MFCCs), it can be observed that there is a pattern in the results. The main reason behind this is that music is a multi-dimensional subject, and the same goes with musical data. There are multiple parameters and factors or traits involved with each music composer that are completely different from the others. Each piece of music sounds completely different. So there are different types of features that are involved in making each musical piece different and distinct. It is difficult for a single deep learning model or single set of features to recognize all the composers distinctly, which can be evidently seen in the results. That is one of the reasons why recognizing a music composer is difficult for even human experts. One of the main reasons why I was able to achieve high accuracy is that different deep learning models and different features tended to recognize some of the composers with high accuracy, leading to overall high accuracy. However, the same deep learning model and features tend to have very low accuracy for some of the composers.

6.1 Future Work

One of the major goals of this research was to implement all possible methodologies and to ascertain what works the best. It will also point all future researchers in this field in the right direction about what to try and what not to try. I have tried to implement all possible features: deep learning models and N-grams models. One of the problems that I faced was getting the right data and a sufficient amount of data.

Even while doing this research, I did not have sufficient amount of data. I feel the results would have been even better if I had more data at hand.

Another thing one can do is to discuss this problem with the stakeholders, such as musicians and composers, in order to understand the unique and distinct traits of each composer. And find a way to represent those features mathematically or in computer-understandable form. A lot of work needs to be done for the right feature representation that would be able to represent all the aspects of the music in totality, so that all possible aspects get covered in those features and nothing gets left out.

The other thing is implementing different deep learning models and parameters and optimization methods to get the optimal results. I implemented different models, and Squeezenet was the one that suited the problem more than others.

Similarly, in the case of N-grams, it was observed that although it provided great results, it was observed that pitch and duration notes are not sufficient parameters for all composers. Although N-grams are a great way to solve this problem, they should be given more parameters rather than just these two features. I think with more parameters, N-grams will give even better results. jSymbolic software can be a great way to achieve the required features as it is capable of extracting 246 audio features. More research needs to be done in order to understand which of these features are relevant with respect to the composers.

Because music is a very ambiguous subject, it is difficult to define it completely. Therefore, a certain set of parameters, features, and models might be sufficient or suitable for certain types of composers. Each composer being completely unique and each having their own set of characteristics, there is a good possibility that two different pieces by the same composer will be totally different. It is not possible to create a universal model or framework for this problem. Having a wholesome approach would be the right approach.

Bibliography

[1] Niko Abeler. Musical composer identification. 2015.

[2] Iffat A Gheyas and Leslie S Smith. Feature subset selection in large dimensionality domains. *Pattern recognition*, 43(1):5–13, 2010.

[3] Nadine Hajj, Maurice Filo, and Mariette Awad. Automated composer recognition for multi-voice piano compositions using rhythmic features, n-grams and modified cortical algorithms. *Complex & Intelligent Systems*, 4(1):55–65, 2018.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Dorien Herremans, David Martens, and Kenneth Sörensen. Composer classification models for music-theory building. In *Computational Music Analysis*, pages 369–392. Springer, 2016.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] María Hontanilla, Carlos Pérez-Sancho, and Jose M Inesta. Modeling musical style with language models for composer recognition. In *Iberian conference on pattern recognition and image analysis*, pages 740–748. Springer, 2013.

[9] Jeremy Howard and Sylvain Gugger. Fastai: a layered api for deep learning. *Information*, 11(2):108, 2020.

[10] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[11] Saahil Jain, Akshay Smit, and Tim Yngesjö. Analysis and classification of symbolic western classical music by composer. *Preprint.[Online]. Available: http://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26583519.pdf.*

[12] Maximos A Kaliakatsos-Papakostas, Michael G Epitropakis, and Michael N Vrahatis. Musical composer identification through probabilistic and feedforward neural networks. In *European Conference on the Applications of Evolutionary Computation*, pages 411–420. Springer, 2010.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[14] Cory McKay and Ichiro Fujinaga. Automatic music classification and the importance of instrument identification. In *Proceedings of the Conference on Interdisciplinary Musicology*, pages 1–10, 2005.

[15] Cory McKay and Ichiro Fujinaga. jsymbolic: A feature extractor for midi files. In *ICMC*, 2006.

[16] Gianluca Micchi. A neural network for composer classification. In *International Society for Music Information Retrieval Conference (ISMIR 2018)*, 2018.

[17] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[18] TJ Tsai and Kevin Ji. Composer style classification of piano sheet music images using language model pretraining. *arXiv preprint arXiv:2007.14587*, 2020.

[19] Harsh Verma and John Thickstun. Convolutional composer classification. *arXiv preprint arXiv:1911.11737*, 2019.

[20] Jacek Wołkowicz, Zbigniew Kulka, and Vlado Kešelj. N-gram-based approach to composer recognition. *Archives of Acoustics*, 33(1):43–55, 2008.

[21] Daniel Yang and Timothy Tsai. Composer classification with cross-modal transfer learning and musically-informed augmentation. In *ISMIR*, pages 802–809, 2021.