UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO ADRIAN PEREIRA MARTINEZ

AVALIAÇÃO DE PERFORMANCE DE ALGORITMOS DE RECONHECIMENTO DE ACORDES MUSICAIS

CAMPINA GRANDE - PB

2024

PEDRO ADRIAN PEREIRA MARTINEZ

AVALIAÇÃO DE PERFORMANCE DE ALGORITMOS DE RECONHECIMENTO DE ACORDES MUSICAIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Leandro Balby Marinho

CAMPINA GRANDE - PB

2024

PEDRO ADRIAN PEREIRA MARTINEZ

AVALIAÇÃO DE PERFORMANCE DE ALGORITMOS DE RECONHECIMENTO DE ACORDES MUSICAIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Leandro Balby Marinho

Orientador – UASC/CEEI/UFCG

Adalberto Cajueiro de Farias

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15/05/2024

CAMPINA GRANDE - PB

RESUMO

No contexto musical, a progressão de acordes é capaz de capturar diversas características de mais alto nível de uma música, como por exemplo os possíveis sentimentos associados, o gênero, a tonalidade da música, o ritmo, etc. No entanto, a identificação da progressão de acordes é um processo que demanda a atuação de especialistas, e é um processo que se torna inviável de ser feito manualmente em grandes bases de dados. Assim, surgiu a necessidade da concepção de classificadores que pudessem extrair essa informação a partir de um sinal sonoro. O objetivo deste trabalho é realizar uma avaliação da performance de algoritmos voltados para esse fim que estão disponibilizados publicamente, verificando quais são as estratégias de aprendizagem utilizadas nos algoritmos que são mais promissoras, assim como identificar as maiores lacunas existentes nesses modelos, de modo a guiar futuros esforços na criação de novos classificadores, e datasets de treinamento.

PERFORMANCE EVALUATION OF CHORD RECOGNITION ALGORITHMS

ABSTRACT

In the musical context, a chord progression is able to capture various higher-level characteristics of a piece of music, such as possible associated emotions, genre, key, rhythm, etc. However, chord progression identification is a process that requires the expertise of specialists and becomes infeasible to be done manually on large databases. Thus, there arose the need for the conception of classifiers that could extract this information from an audio signal. The objective of this work is to evaluate the performance of algorithms aimed at this purpose that are publicly available, verifying which learning strategies used in the algorithms are the most promising, as well as identifying the major gaps in these models, in order to guide future efforts in creating new classifiers and training datasets.

Avaliação de performance de algoritmos de detecção de acordes musicais

Pedro Adrian Pereira Martinez
Universidade Federal de Campina Grande
pedro.martinez@ccc.ufcg.edu.br

RESUMO

No contexto musical, a progressão de acordes é capaz de capturar diversas características de mais alto nível de uma música, como por exemplo os possíveis sentimentos associados, o gênero, a tonalidade da música, o ritmo, etc. No entanto, a identificação da progressão de acordes é um processo que demanda a atuação de especialistas, e é um processo que se torna inviável de ser feito manualmente em grandes bases de dados. Assim, surgiu a necessidade da concepção de classificadores que pudessem extrair essa informação a partir de um sinal sonoro. O objetivo deste trabalho é realizar uma avaliação da performance de algoritmos voltados para esse fim que estão disponibilizados publicamente, verificando quais são as estratégias de aprendizagem utilizadas nos algoritmos que são mais promissoras, assim como identificar as maiores lacunas existentes nesses modelos, de modo a guiar futuros esforços na criação de novos classificadores, e datasets de treinamento.

Palavras-chave

Reconhecimento de acordes, aprendizado de máquina, processamento de áudio, harmonia musical.

1. INTRODUÇÃO

Quando consideramos o contexto de recuperação de informações musicais, estamos interessados em obter de forma automática features que descrevam em mais alto nível uma faixa musical, a partir do sinal de áudio bruto. Como exemplo de features associadas a uma música, temos o seu gênero, o ritmo, o andamento, a progressão de acordes presente na música, quais sentimentos podem estar associados à mesma, entre outras possibilidades.

Das features citadas, uma que apresenta grande importância no contexto anteriormente mencionado é a extração da progressão de acordes, uma vez que captura aspectos estruturais de uma música de uma maneira compacta, assim atraindo a atenção em aplicações musicais [9]. A detecção ou reconhecimento de acordes musicais é o processo de atribuir labels de acordes para diferentes segmentos de uma determinada faixa de áudio. Algoritmos que realizam essa tarefa possuem uma gama variada de aplicações, dentre elas podemos citar a extração de metadados de faixas musicais, que então podem ser utilizados em sistemas de recomendação, ou ainda a transcrição automática de músicas em partituras, e para fins educacionais [1].

A extração da progressão de acordes a partir de uma faixa musical é um tópico ainda com muita discussão e espaço para realização de pesquisas, uma vez que o contexto associado à essa tarefa possui algumas complexidades inerentes, tanto em relação à construção de um classificador que seja preciso, assim como na definição do *ground truth* de bases de dados, uma vez que essa definição requer expertise, ao contrário de tarefas como reconhecimento de objetos, o que leva à pouca disponibilidade de datasets públicos, e por consequência disso, leva a problemas como classificadores enviesados [2].

Dadas as lacunas anteriormente mencionadas, surge o interesse em avaliar a performance de algoritmos de reconhecimento de acordes já existentes, pois assim podemos observar quais são os tipos de erros mais comuns cometidos pelos modelos, o que poderia guiar futuros esforços na concepção de novos classificadores, assim como na construção de novos datasets públicos, que apresentem maior variedade de acordes, e assim reduzindo o enviesamento dos classificadores apontado previamente.

Um dos objetivos deste trabalho é realizar um comparativo da performance de diferentes algoritmos de detecção de acordes, de forma quantitativa, a partir de métricas como precisão e F1-score. Com essa comparação quantitativa, poderíamos ter uma ideia de quais são as estratégias adotadas nos algoritmos de reconhecimento de acordes que são mais promissoras. Outro objetivo é realizar uma análise crítica de tais algoritmos, investigando quais são as lacunas presentes e os cenários nos quais apresentaram maior taxa de erros. Os algoritmos foram avaliados a partir de um dataset com 909 músicas instrumentais livres de direitos autorais [3].

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Fundamentos de teoria musical

Antes de nos aprofundarmos nos fundamentos dos algoritmos de reconhecimento de acordes, é importante introduzirmos alguns conceitos sobre as estruturas musicais. As notas são os menores elementos constituintes da música. Ao representarmos as notas, é comum utilizarmos uma notação que consiste em uma letra, representando o nome da nota, também conhecido por qualidade ou croma da nota, seguido por um número, que representa a oitava da nota (o conceito de oitava será discutido em seguida). Por exemplo, a nota de afinação para uma orquestra sinfônica é denotada pela notação A4, onde A representa a nota Lá, e 4 é o número da oitava [4].

Na música ocidental, existem as chamadas notas naturais, que são sete, e são representadas pelas letras A, B, C, D, E, F, G. Essas notas naturais correspondem às teclas brancas de um piano. Um intervalo é a distância entre duas notas. Dizemos que o intervalo

entre duas notas naturais consecutivas é de um tom , com exceção do intervalo entre as notas Si(B) e Dó(C) , e Mi(E) e Fá(F), que nesses casos é de um semitom. Se considerarmos todas as notas no sistema musical ocidental, temos 12 notas, que são espaçadas por intervalos de semitom, que podem ser denotadas da seguinte forma : A A# B C C# D D# E F F# G G#. Essa sequência de notas é conhecida como escala cromática, e será referenciada mais adiante.

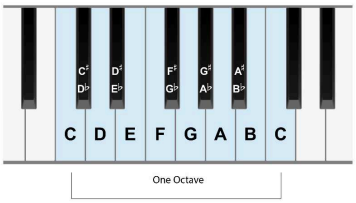

Figura 1 - Notas presentes em uma oitava de um piano. Fonte: [6]

Um conceito importante para termos em mente é a altura de uma nota. Definimos uma altura de uma nota como sendo a percepção humana de uma onda sonora em uma frequência específica [4]. Assim, a altura da nota A4 corresponde à uma frequência de 440 Hz.

Porém, temos que ter em mente que na natureza, ondas sonoras musicais não oscilam seguindo apenas uma única frequência. Quando, por exemplo, oscilamos uma corda de um violão para produzir um som, além da frequência principal da nota, que chamamos de altura, também são produzidos os chamados harmônicos, que são frequências correspondentes à multiplicação da frequência base por números inteiros [1].

Definimos uma escala musical como sendo uma sequência de notas com uma estrutura de intervalos bem definida. Conforme seguimos as alturas de uma escala de baixo para cima, começamos e finalizamos na mesma nota. Nesse caso dizemos que percorremos um intervalo de uma oitava. Notas com uma oitava de diferença soam semelhantes ao ouvido humano porque um salto de uma oitava corresponde a uma duplicação na frequência da onda sonora [4].

A escala cromática, que foi mostrada anteriormente, foi definida com uma propriedade interessante, na qual as 12 notas são espaçadas quase perfeitamente de forma logarítmica ao longo da oitava. Já que uma nota uma oitava acima possui o dobro da frequência, e utilizando essa propriedade mencionada, podemos usar uma sequência recursiva para descrever a escala cromática [4]:

$$P_i = 2^{1/12} P_{i-1}$$

Onde P_i é a frequência de uma i-ésima nota qualquer em relação à frequência da nota precedente, denotada por P_{i-1}. Como foi discutido anteriormente, o intervalo entre duas notas consecutivas da escala cromática é um intervalo de semitom (S). Já o intervalo de dois semitons é chamado de tom (T).

Com tais conceitos em mente, podemos definir duas escalas bastante utilizadas na música ocidental, que são a escala maior e menor, a partir das seguintes sequências de tons e semitons, a partir de uma nota inicial, chamada de tônica. Com tais sequências de intervalos, é garantido que começamos e terminamos na mesma nota, espaçada por um intervalo de uma oitava.

Escala	Sequência de Intervalos
Escala Maior	TTSTTTS
Escala Menor	TSTTSTT

Tabela 1 - Definição das escalas maior e menor a partir da sequência de intervalos em relação à nota tônica

Múltiplas notas tocadas simultaneamente são definidas como um acorde. Um acorde de tríade é um acorde simples que contém a primeira, a terceira e a quinta nota de uma escala. Por exemplo, uma tríade de Dó maior contém a primeira , terceira e quinta nota da escala de Dó maior, ou seja, Dó, Mi e Sol. Já um acorde de tétrade contém uma nota adicional, que é a sétima nota da escala. Um exemplo de acorde desse tipo é o acorde de sétima menor, denotado por min7, formado a partir da tríade menor, com a adição da sétima nota da escala menor.

Acordes também podem ser formados a partir da adição de notas que não compõem a estrutura básica de um acorde, isto é, as notas da tétrade. Tais notas são chamadas de extensões de acordes. Ou seja, as extensões possíveis são a segunda , quarta e sexta nota de uma escala, que também podem ser chamadas de nona, décima primeira, e décima terceira, caso estejam em uma oitava acima da tônica. Por fim, também existe o conceito de inversão de acordes. Uma inversão é a troca da nota com a frequência mais grave do acorde, chamada de baixo, que normalmente deveria ser a tônica, por alguma outra nota presente no acorde. Um exemplo de inversão é a chamada primeira inversão de Dó maior, que corresponde a um acorde de Dó maior com a terça maior, ou seja, a nota Mi, no baixo. Esse acorde, na notação MIREX[1], que é a notação normalmente utilizada nos trabalhos relacionados a tarefas de recuperação de informações musicais, assim como nos algoritmos de reconhecimento de acorde, é denotado por C:maj/3.

2.2 Processamento de sinais sonoros

2.2.1 Introdução

Uma parte crucial de um modelo de detecção de acordes é a análise do sinal sonoro. Um sinal sonoro bruto nos dá pouca informação, pois estamos mais interessados em quais frequências compõem aquele sinal sonoro. A principal ferramenta matemática usada para decompor o sinal de áudio bruto é a Transformada Rápida de Fourier (FFT), que é um algoritmo otimizado da transformada discreta de Fourier. Utilizamos a FFT para determinar as alturas que estão presentes em nosso sinal bruto [4].

2.2.2 Introdução à transformada de Fourier

A transformada de Fourier foi nomeada em homenagem ao matemático francês Jean-Baptiste Joseph Fourier. Ele introduziu a série de Fourier no início do século XIX como parte de seu trabalho sobre a condução de calor, no livro “La Théorie Analitique de la Chaleur”. Neste trabalho, Fourier demonstrou que qualquer função periódica, por mais complicada que pareça, pode ser expressa como uma soma de senos e cossenos. A

[1]https://music-ir.org/mirex/wiki/2021:Audio_Chord_Estimation

Transformada de Fourier, que generaliza a série de Fourier para funções não periódicas, foi formalizada mais tarde [5].

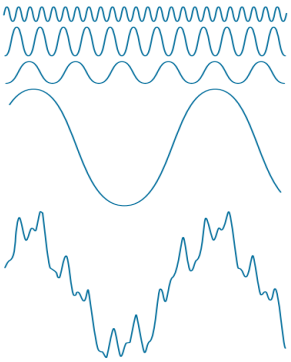

Figura 2 - Soma de várias ondas senoidais resultando em uma onda mais complexa. Fonte: [5]

A Transformada de Fourier é uma ferramenta matemática que desmembra uma função de tempo f(t) em suas componentes de frequência. Ou seja, ela nos permite ver quais frequências estão presentes em nosso sinal e a magnitude das mesmas. Essa transformação é crucial para realizar a análise de sinais, já que auxilia na compreensão de como diferentes frequências contribuem para uma forma de onda complexa. A transformada de Fourier é reversível, com a transformada inversa de Fourier retornando a função no domínio do tempo a partir de uma função no domínio da frequência [5].

Sinais de áudio são gravados e ouvidos no domínio do tempo, ou seja eles representam a intensidade do som como uma função do tempo. No entanto, alturas e, portanto, acordes são representados por frequências. Portanto, usamos a transformada de Fourier para converter o sinal de entrada do domínio do tempo em uma representação de frequência que pode ser analisada quanto às intensidades de frequências específicas. A relação entre as frequências em um dado momento pode ser então analisada para determinar o acorde que está sendo tocado [4].

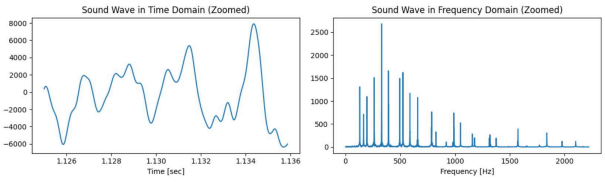

Figura 3 - Uma onda sonora, representada no domínio do tempo, à esquerda, e no domínio da frequência, à direita. Fonte: Autoria própria.

2.2.3 *Transformada de Fourier contínua*

A transformada de Fourier é uma ferramenta matemática que é usada para transformar uma função no domínio do tempo f(t) para o domínio da frequência $F(\mu)$. A transformada de Fourier é uma transformação no plano complexo. Ou seja, mesmo para entradas de valores reais, como sinais de áudio, ela retorna uma representação no domínio da frequência de valores complexos [1]. Definimos a transformada de Fourier de f(t) como sendo [5]:

$$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \mu t} dt$$

Onde μ também é uma variável contínua. Tendo $F(\mu)$ podemos obter f(t) de volta, realizando a transformada inversa, que é dada por [5]:

$$f(t) = \int_{-\infty}^{\infty} F(\mu) e^{2\pi i \mu t} d\mu$$

Essas equações fornecem uma base teórica para os algoritmos que serão utilizados para encontrar os espectros das ondas, mas como os sinais digitais não são contínuos, precisamos encontrar uma maneira de aplicar a Transformada de Fourier em funções discretas em vez de funções contínuas. Assim, somos motivados a definir a transformada discreta de Fourier.

2.2.4 *Transformada de Fourier discreta (DFT)*

Quando uma onda sonora analógica é armazenada em um computador, ela passa pelo processo de amostragem, que é a transformação de uma função de onda contínua para uma função discreta [4], ou seja, é selecionado um número finito de pontos da curva. Dependendo da quantidade de bits utilizados, é possível tornar a amostragem mais ou menos precisa. Pensando nesse caso discreto é que foi formulada a transformada de Fourier discreta (DFT). Definimos a DFT sobre um vetor complexo f_n de comprimento n como sendo dada por [5]:

$$\widehat{F}(\mu) = \sum_{n=-\infty}^{\infty} f_n e^{-2\pi i \mu n \Delta T}$$

Onde ΔT é o intervalo de tempo entre as amostragens. A DFT inversa possui um formato semelhante ao da equação acima. É possível mostrar a derivação das equações da DFT a partir da definição da transformada de Fourier contínua por meio da chamada função delta de Dirac. A função delta de Dirac é uma função nula em todo ponto, exceto em zero, onde tem valor infinito. Podemos deslocar a função delta para que esteja centrada ao redor de qualquer ponto no eixo x, e não apenas na origem. Essa função tem propriedades matemáticas interessantes, que são então exploradas para discretizar integrais [4]. Existe um conceito denominado de trem de impulsos, que correspondem a funções delta igualmente espaçadas no tempo. Com esse conceito, uma função analógica pode ser amostrada da seguinte forma, como mostra a figura a seguir.

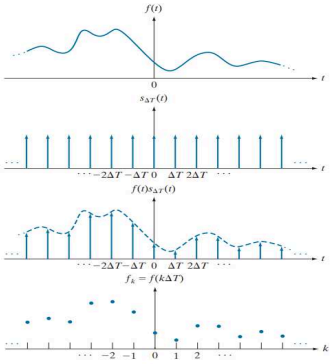

Figura 4 - Etapas para realizar a amostragem de um sinal contínuo. Fonte: [5]

Primeiro, temos a função contínua que queremos amostrar. Em seguida, é feito o produto da função por um trem de impulsos, que está representado no segundo gráfico. Após isso realizamos a integração desse produto, e obtemos a função amostrada, mostrada no último gráfico [5]. É possível mostrar a derivação das equações da DFT ao realizarmos a Transformada de Fourier de um trem de impulsos genérico.

2.2.5 Transformada rápida de Fourier (FFT)

Em 1965, James Cooley e John Tukey publicaram um algoritmo que mudou significativamente o cenário de processamento de sinais digitais. Ao explorar simetrias da DFT, eles conseguiram reduzir o tempo de execução da DFT de $O(n^2)$ para $O(n\ log_2 n)$. Esse algoritmo é a Transformada Rápida de Fourier (FFT), nomeada por esse aumento na velocidade computacional [4].

2.2.6 Transformada de Fourier de curta duração (STFT)

A Transformada de Fourier fornece informações sobre a distribuição de frequências considerando todo o sinal. Na prática, muitas vezes estamos interessados em como a distribuição de frequências muda ao longo de janelas de tempo menores, por exemplo ao analisar a progressão de acordes de uma música. Uma maneira de superar essa limitação é não transformar o sinal inteiro de uma vez, mas sim calcular a transformada para pequenos intervalos de tempo consecutivos independentemente por meio de uma janela deslizante. Essa abordagem é chamada de Transformada de Fourier de curto tempo (STFT). A STFT é definida como [1]:

$$STFT(r,m) = \sum_{k=0}^{N-1} c(r.I - \frac{N}{2} + k) + \omega(\frac{2k}{N} - 1).e^{\frac{-2\pi ikm}{N}}$$

A STFT é função do passo de tempo r , e de m, que corresponde a faixa de frequência (*frequency bin*), que é o tamanho dos intervalos que dividem o espaço de frequências. *I* denota a distância entre dois passos de tempo consecutivos, e ω é uma função de janela que desliza sobre o sinal. Alguns tipos possíveis de função de janela são a retangular, triangular, Hamming, Hann, entre outras [1].

Além de escolher uma função de janela adequada, há um compromisso a ser feito entre a resolução temporal e a de frequência ao escolher o tamanho da janela. Janelas maiores aumentam a densidade de amostragem de frequência, mas, ao mesmo tempo, implicam em uma amostragem mais esparsa no domínio do tempo.

Por outro lado, tamanhos de janela menores podem fornecer uma boa precisão temporal, mas podem ter uma resolução pobre no domínio da frequência [1].

2.2.7 Espectrograma

O espectro de uma onda sonora, produzido pela FFT ou STFT, pode ser visualizado em um diagrama bi-dimensional chamado de espectrograma, cujo eixo x é linear e corresponde ao tempo, e cujo eixo y, que pode ser logarítmico ou linear, corresponde à frequência. As cores associadas às frequências representam as amplitudes das mesmas.

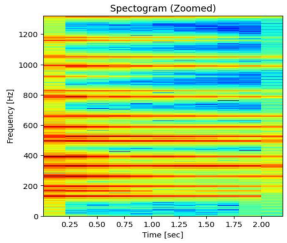

Figura 5 - Espectrograma de uma faixa de áudio. Fonte: Autoria própria

2.2.8 Cromagrama

Em um dos trabalhos pioneiros na área de reconhecimento de acordes [10], foi introduzido o conceito de PCP (pitch-class profile), que representa a distribuição de energia das frequências presentes em um sinal em relação às frequências associadas às 12 classes de notas da escala cromática. A sequência temporal do vetor PCP é chamada de cromagrama [9]. Ao enxergarmos um cromagrama graficamente, o mesmo possui um formato semelhante ao espectrograma, que foi discutido na seção anterior, com a diferença de que no lugar de termos a frequência em valor absoluto no eixo y, temos as notas da escala cromática. Assim, essa representação é de mais alto nível quando comparada ao espectrograma, e mais adequada para a análise musical.

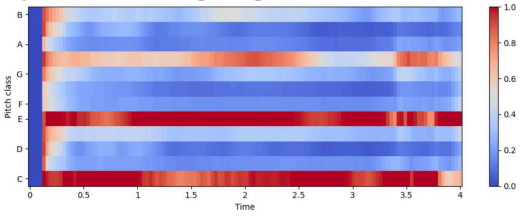

Figura 6 - Visualização gráfica de um cromagrama, para uma faixa de áudio contendo um acorde. Fonte: Autoria própria

2.3 Fundamentos em aprendizagem de máquina

2.3.1 Introdução

Algoritmos de reconhecimento de acordes podem utilizar diferentes estratégias na etapa de correspondência de padrões, que será discutida na seção 2.4. Tais estratégias são recorrentes em diversos problemas relacionados à aprendizagem de máquina, sendo assim crucial entendermos conceitos básicos relacionados a esses modelos antes de entendermos como funciona um algoritmo de reconhecimento de acordes. A estratégia mais comum utilizada baseia-se na utilização de um modelo oculto de Markov (HMM), que é um modelo estatístico. Recentemente, muitos estudos têm explorado redes neurais profundas, como redes neurais convolucionais (CNNs) ou redes neurais recorrentes (RNNs) para o reconhecimento de acordes [7]. Outra tendência observada na área de recuperação de informações musicais é a utilização de arquiteturas transformers [14].

2.3.2 Modelo oculto de Markov (HMM)

Uma cadeia de Markov é um modelo estatístico no qual temos uma sequência de eventos, em que a probabilidade de um evento futuro depende apenas do evento atual. Isso é chamado de propriedade de Markov. Em outras palavras, o estado presente de um sistema é o único que determina seu estado futuro,

independentemente de como esse estado foi alcançado [11]. Uma cadeia de Markov pode ser representada por um diagrama de estados. Nesse diagrama, os nós representam cada estado possível do sistema, e setas ou arestas representam as transições entre esses estados. Cada transição está ligada a uma probabilidade, que representa a chance de ocorrer a transição de um estado para outro.

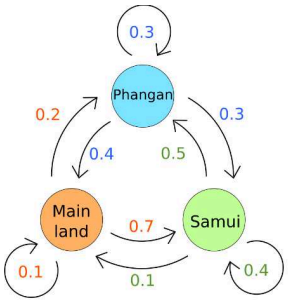

Figura 7 - Cadeia de Markov usada para modelar o roteiro de viagens de um turista na Tailândia. Fonte: [12]

Um modelo oculto de Markov (HMM) é uma extensão do conceito de cadeia de Markov. Nele, temos variáveis que são observadas, cujos valores observados são dependentes de uma cadeia de Markov, cujo estado real não é diretamente acessível. Esse modelo é aplicado em diversos contextos, como processamento de linguagem natural, modelos de reconhecimento de fala, visão computacional, entre outros. Em um modelo de reconhecimento de fala, por exemplo, as palavras faladas são os estados ocultos e a variável observada é a forma de onda de áudio que é capturada por um microfone. Assim, o HMM determina a sequência mais provável de palavras correspondentes às observações de áudio.

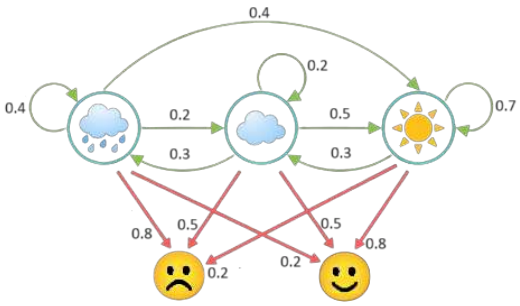

Figura 8 - Modelo oculto de Markov, onde o humor de uma pessoa é dependente do clima do dia, que é modelado por uma cadeia de Markov. Fonte: Autoria própria

Podemos entender como um modelo desse tipo se encaixa no contexto de reconhecimento de acordes, uma vez que a progressão de acordes de uma música poderia ser modelada como uma cadeia de Markov, já que os acordes que estão dentro da tonalidade de uma música normalmente possuem funções harmônicas bem definidas, e assim por exemplo, um acorde que produz uma sensação de tensão na música provavelmente será sucedido por um acorde com sensação de resolução. No entanto, essa cadeia seria desconhecida a priori, e teríamos como variável observada apenas o cromagrama gerado a partir da faixa de áudio.

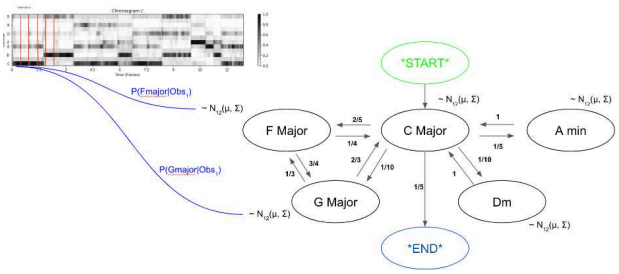

Figura 9 - Aplicação de um modelo oculto de Markov no contexto de reconhecimento de acordes. Fonte: [13]

2.3.3 Redes neurais profundas

2.3.3.1 Introdução

No contexto de aprendizagem de máquina, uma rede neural artificial (ANN) é um modelo inspirado pela estrutura de redes neurais biológicas em cérebros animais. Uma ANN consiste em nós, que são chamados de neurônios, conectados por arestas, que modelam as sinapses em neurônios biológicos. Cada aresta possui um valor numérico associado, que modela a força de uma determinada conexão [15]. Cada neurônio de uma rede neural recebe como entrada os sinais de outros neurônios (onde cada sinal é um número real) multiplicados pelos pesos das arestas que os conectam ao neurônio atual, e então por meio de uma função computada sobre a soma de suas entradas, chamada função de ativação, produz sua saída [16]. O processo de treinamento de uma rede neural envolve ajustar os pesos das conexões entre os neurônios de modo que a rede produza saídas desejadas para um conjunto de entradas de treinamento. Esse ajuste é feito iterativamente, por meio de algoritmos como gradiente descendente e *back propagation* [5].

Neurônios se encontram agregados em camadas. A primeira camada é chamada de camada de entrada, e a última camada é a camada de saída. As camadas intermediárias são chamadas de camadas ocultas. Uma rede neural é chamada de rede neural profunda se possuir 3 ou mais camadas intermediárias, no entanto, na prática costuma ter muito mais camadas [17].

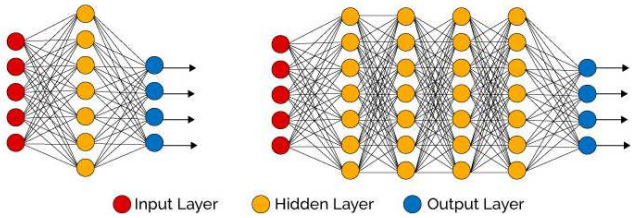

Figura 10 - Comparação entre uma rede neural simples e uma rede neural profunda. Fonte: [18]

Algumas limitações surgiram durante a aplicação das primeiras redes neurais, que foram conceituadas nas décadas de 1940 a 1960, como o fato de não conseguirem aprender funções relativamente simples, como a função lógica XOR [14]. Com o advento de maneiras eficientes de treinar redes neurais, houve a viabilização da utilização de redes neurais profundas, e assim tais limitações foram superadas, e motivou a aplicação de redes neurais profundas nos mais diversos campos do conhecimento [14].

2.3.3.2 Redes neurais convolucionais

Em uma rede neural convencional, como um perceptron multicamadas, cada neurônio da rede está conectado a outro neurônio da próxima camada por meio de um parâmetro único, que é o peso da aresta. Isso permite que a rede seja mais expressiva e modele problemas mais complexos. No entanto, isso causa uma explosão combinatória com o aumento de camadas e neurônios [14]. As redes neurais convolucionais (CNNs), inspiradas no sistema visual biológico, promovem uma reutilização de parâmetros ao longo da rede, reduzindo assim a complexidade computacional do treinamento da rede. É comum nos referirmos aos neurônios que compartilham o mesmo parâmetro como o campo receptivo do parâmetro, um termo retirado da neuropsicologia [14].

Os parâmetros compartilhados são modelados a partir das camadas convolucionais. As camadas convolucionais consistem em filtros (kernels) que são aplicados à entrada para extrair padrões de características. Cada filtro percorre a entrada e realiza operações matemáticas para produzir um mapa de características. Esses mapas de características podem ser interpretados como representações intermediárias que capturam informações de nível mais alto, como por exemplo bordas em imagens, e assim são utilizados para reduzir a dimensionalidade da entrada. Dado o funcionamento dos kernels de convolução, as CNNs têm sido uma técnica útil para lidar com estruturas fixas de comprimento determinado, em formato de grade (por exemplo, imagens), produzindo resultados positivos em muitas tarefas [14]. Na área de extração de informações musicais, CNNs foram aplicadas com sucesso em tarefas como reconhecimento de gêneros musicais, reconhecimento de acordes, estimativa de tonalidade, entre outras [14].

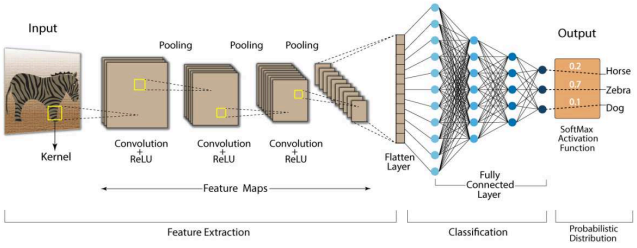

Figura 11 - Arquitetura de uma rede neural convolucional. Fonte: [20]

2.3.3.2 Redes neurais recorrentes

As Redes Neurais Recorrentes (RNNs) são um tipo de redes neurais projetadas para lidar com dados sequenciais. Ao contrário da maioria das outras arquiteturas de redes neurais, as RNNs não assumem que as entradas são independentes entre si. Em vez disso, elas atualizam seus parâmetros considerando não apenas o input atual para a rede, mas também os inputs anteriores processados pela rede [14]. As RNNs foram conceituadas com o surgimento do algoritmo de *back propagation,* no entanto a dificuldade de treinamento das arquiteturas iniciais impedia o uso de RNNs em aplicações práticas. Tal dificuldade foi superada com o surgimento da arquitetura *Long Short-Term Memory* (LSTM) [14]. Na área de extração de informações musicais, RNNs foram aplicadas com sucesso em tarefas como geração de músicas, transcrição de músicas, e reconhecimento de acordes [14].

2.3.3.2 Arquitetura transformers

Uma rede com arquitetura transformer é um modelo sequencial introduzido por [19]. O transformer difere de outras arquiteturas sequenciais, pois depende inteiramente de um conceito chamado de mecanismo de atenção (que foi primeiramente implementado em modelos sequenciais baseados em RNNs) para inferir dependências globais entre a entrada e saída, ou seja, não inclui arquiteturas recorrentes ou convolucionais [7]. O mecanismo de atenção permite ao modelo concentrar sua atenção em partes específicas de uma entrada. Isso é útil em tarefas onde a relação entre diferentes partes da entrada é importante para a saída do modelo. Com o uso da arquitetura transformers, foi possível mitigar as limitações de modelos sequenciais anteriores. Em tais modelos, o desempenho se degrada à medida que o comprimento da sequência aumenta [14]. A formulação matemática da função de atenção é [19]:

$$Attention(K, Q, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Onde K, Q e V são vetores referenciados referenciados por [19] como *key, query,* e *value*, e d_k é a dimensão do vetor K. De maneira simplificada, o vetor Q é uma representação de uma consulta que o modelo está realizando na entrada. Em relação ao vetor K, tem como finalidade ajudar o modelo a determinar quais partes da entrada são mais importantes em relação à consulta. Já em relação ao vetor V, é a representação da entrada em um formato que o modelo deseja para assim produzir a saída.

2.4 Algoritmos de reconhecimento de acorde

A maioria dos sistemas tradicionais de reconhecimento automático de acordes consiste em três partes: extração de características de áudio, correspondência de padrões (*pattern matching*) e decodificação da sequência de acordes, também chamado de pós-filtragem [8]. Além disso, também pode conter uma etapa de pré-filtragem [9].

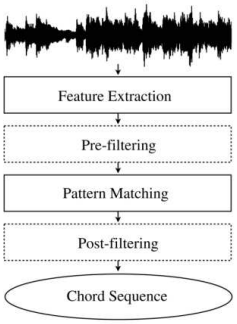

Figura 12 - Arquitetura básica de um sistema de reconhecimento de acordes. Fonte: [9]

Na etapa de extração de características, são extraídas as informações harmônicas a partir de um sinal de áudio. Isto corresponde à extração do cromagrama, que foi discutido na seção 2.2.8. Em relação à pré-filtragem, ela é aplicada diretamente aos cromagramas antes da etapa de correspondência de padrões. Nesta etapa, pode-se suavizar janelas ruidosas, por meio da aplicação de filtros baseados na média ou mediana [9]. Devido à prevalência do uso da abordagem baseada em PCP na etapa de extração de

características, o que torna cada sistema de reconhecimento de acordes único é o sistema de decisão, utilizado na etapa de correspondência de padrões [9]. É nesta etapa que entram os modelos discutidos na seção 2.3.

A etapa de pós-filtragem, que segue a etapa de correspondência de padrões, é normalmente aplicada no contexto de sistemas baseados em HMMs, com algumas poucas exceções. Basicamente consiste na aplicação de um decodificador de Viterbi, que encontra a sequência de acordes mais provável ao avaliar a probabilidade conjunta, que foi obtida na etapa de correspondência de padrões, juntamente com todas as possíveis sequências de acordes com base em uma matriz de transição de estados. Em outras palavras, ele reduz o número de previsões incorretas restringindo transições de acordes improváveis [9].

3. METODOLOGIA

3.1 Escolha do dataset

A avaliação dos modelos de reconhecimento de acordes foi feita a partir da utilização do dataset POP 909 [3], que é um dataset com 909 músicas em formato MIDI, juntamente com arquivos de texto com anotações a respeito do tempo das músicas, tonalidade, e com anotações dos acordes presentes nas faixas de áudio, que é a parte de interesse. A escolha do dataset se deu principalmente pelo fato de que os arquivos de áudio de referência foram disponibilizados publicamente, o que infelizmente não é comum em datasets desse tipo, principalmente por questões de direitos autorais. Assim, cada modelo foi executado tendo como entrada as faixas de áudio do dataset, e então foi feita a comparação das anotações de acordes (ground truth), que foram extraídas do dataset, com as anotações previstas pelos modelos. Para a execução dos modelos, foi necessário antes realizar a geração dos arquivos de áudio, em formato WAV ou MP3, a partir dos arquivos MIDI, uma vez que arquivos MIDI são arquivos cujo conteúdo inclui as notas a serem tocadas, seu tempo, duração e volume desejado para cada nota, não sendo um arquivo de áudio propriamente dito. A geração dos arquivos de áudio pode ser feita através da utilização de plugins e ferramentas como o ffmpeg.

3.2 Escolha dos modelos a serem avaliados

A metodologia adotada para escolher os modelos de reconhecimento de acordes a serem avaliados envolveu uma pesquisa em repositórios de código públicos, com foco nos sites Papers with Code[2] e Github. O processo de seleção foi feito a partir da exploração de repositórios por tema, especificamente buscando por repositórios relacionados ao reconhecimento de acordes. Um dos critérios de seleção adotados foi a exigência de que a solução tivesse a capacidade de processar arquivos de áudio diretamente. Algumas soluções foram descartadas devido a esse critério, uma vez que as mesmas se destinavam exclusivamente ao reconhecimento de acordes em tempo real por meio de uma entrada de áudio. Além disso, outros critérios levados em conta para o descarte de soluções foram a baixa performance e a dificuldade de utilização, principalmente por documentação insuficiente. As tabelas a seguir sumarizam os resultados obtidos na pesquisa por soluções adequadas.

Repositório	Aceitação	Motivo de descarte
Melody transcription via generative pre-training	Descartado	Produz uma saída em formato muito diferente do esperado
AugmentedNet: A Roman Numeral Analysis Network	Descartado	Tem relação com o tema, mas tem outra finalidade
Omnizart: A General Toolbox for Automatic Music Transcription	Aceito	
A Bi-directional Transformer for Musical Chord Recognition	Aceito	
Feature Learning for Chord Recognition: The Deep Chroma Extractor	Descartado	Falta de documentação no código associado
Chord Recognition in Symbolic Music: A Segmental CRF Model	Descartado	Processa arquivos em formato MusicXML, que não é o caso do dataset utilizado
Explaining Deep Convolutional Neural Networks on Music Classification	Descartado	O código associado tem outra finalidade

Tabela 2 - Análise de repositórios presentes no site Papers with code.

Repositório	Aceitação	Motivo de descarte
cjbayron / autochord	Aceito	
orchidas/ Chord-Recognition	Descartado	Realiza predições apenas em formato de tríades, performance baixa em testes preliminares

[2] https://paperswithcode.com

christofw / pitchclass_mctc	Descartado	Tem apenas exemplos de códigos que são referenciados em um artigo
Minichain / EversongApp	Descartado	Não fornece a possibilidade de processar arquivos de áudio
stackswithans / akkorder	Descartado	Não ficou claro como realizar o passo 3 nas instruções para obtenção do cromagrama
derrickward / ChordRecGen	Descartado	Biblioteca para aplicativos mobile para detectar um único acorde a partir de um arquivo MIDI
aisu-programming / Chord-Recognition	Descartado	Performance não promissora, projeto apenas para uso pessoal do autor
zukarusan / JChoreco	Descartado	Realiza predições apenas em formato de tríades
arulandu / chordy	Descartado	Não fornece a possibilidade de processar arquivos de áudio
christofw / pitchclass_cnn	Descartado	Tem apenas exemplos de códigos que são referenciados em um artigo
ohollo/ chord-extractor	Aceito	
Tolu1 / chordler	Descartado	Projeto ainda em desenvolvimento

Tabela 3 - Análise de repositórios presentes no Github.

Desconsiderando as soluções descartadas, restaram assim 4 soluções adequadas para nossa análise: Chordino, Autochord, Omnizart e BTC-ISMIR19. Em relação a primeira solução, foi necessário criar uma função para gerar os arquivos de textos com as predições do modelo para uma música. Além disso, também foi necessário transformar a notação de acordes presentes nos arquivos de textos gerados, para que ficassem em um formato igual ao dos outros modelos, assim como do dataset. Em relação ao Autochord, houve uma certa dificuldade em executá-lo, uma vez que utilizava algumas funções obsoletas, e assim foi feito um *downgrade* de algumas das dependências, pois seria mais fácil do que adaptar o código. Houve dificuldade de instalação do Omnizart, e assim optou-se pela utilização da imagem Docker que é disponibilizada no repositório.

3.3 Avaliação da performance dos modelos

Os arquivos com as anotações dos acordes de uma faixa de áudio, tanto do dataset, como os gerados pelos modelos de reconhecimento, são arquivos de texto que possuem o seguinte formato: Em cada linha temos 3 valores separados por espaço, onde o primeiro e o segundo valor representam o tempo inicial e final de um intervalo de tempo, e o terceiro valor é a anotação de acorde associada àquele intervalo de tempo. Para ilustrar como as métricas de performance dos modelos foram calculadas, vamos considerar um exemplo, onde temos na figura a seguir a linha do tempo de uma faixa de áudio como descrita pelo dataset, ou seja, com as anotações de acordes esperadas em cada intervalo de tempo. Essa linha do tempo está representada pelo gráfico azulado, já a linha do tempo descrita pelo algoritmo de reconhecimento está representada pelo gráfico amarelo.

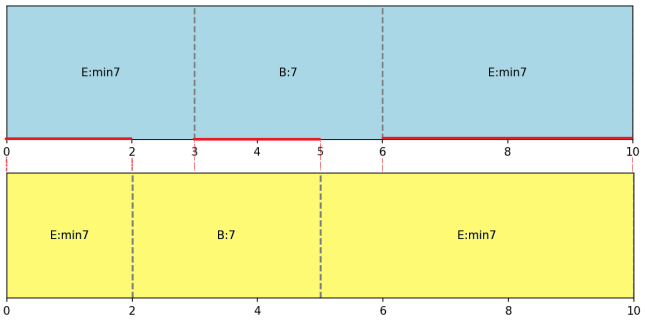

Figura 13 - Ilustração do cálculo de precisão. Fonte: Autoria própria

A linha vermelha acima do eixo x do gráfico azul, que é o tempo em segundos, representa as regiões onde houve acertos do algoritmo. Vemos que em tais regiões o algoritmo previu os acordes corretamente. No entanto, existem regiões que não foram cobertas pelo algoritmo. Por exemplo, no instante de tempo de 2 a 3 segundos, o acorde esperado era E:min7, enquanto que o acorde previsto neste instante de tempo pelo algoritmo foi B:7. Adaptando as definições de precisão e recall para a situação, chegamos à seguinte formulação:

$$Recall = \frac{Região\ total\ de\ acertos}{Região\ correta\ total}$$

$$Precisão = \frac{Região\ total\ de\ acertos}{Região\ prevista\ total}$$

A definição de Recall formulada é equivalente à métrica que é referida na literatura como *overlap ratio* (OR) [1]. Nesse exemplo, o recall do algoritmo foi de 80%, uma vez que no total 8 segundos da linha de tempo descrita pelo dataset foram cobertos.

A precisão nesse caso também foi de 80%, já que o tamanho da linha do tempo prevista é igual ao tamanho da linha do tempo descrita pelo dataset. Foi projetado um algoritmo[3] para calcular as métricas de recall e precisão a partir da verificação das linhas de tempo mencionadas. O algoritmo realiza uma ordenação dos intervalos de tempo presentes em ambas as linhas de tempo, baseando-se no valor de tempo final, para assim achar os instantes de tempo relevantes, e então itera sobre os mesmos e realiza as verificações.

Para comparar se dois acordes são equivalentes, não é simplesmente feita a comparação entre as strings. Isso foi pensado para permitir uma maior flexibilidade na nossa análise, já que existe a possibilidade de que uma nota tônica de um acorde possa ser denotada de duas formas diferentes, fenômeno conhecido como enarmonia. Como exemplo, podemos citar as notas Fá sustenido (F#) e Sol bemol (Gb). Além disso, podemos querer flexibilizar os acertos do algoritmo em relação às extensões esperadas dos acordes. Relembrando o que são extensões de acordes, são notas que compõem um acorde mas que não fazem parte da estrutura básica do mesmo. Em relação a este último ponto, foi utilizado o coeficiente de Jaccard para fazer uma análise da similaridade de notas, incluindo as extensões, entre dois acordes. A definição do coeficiente de Jaccard, para dois conjuntos A e B é a seguinte:

$$J = \frac{|A \cap B|}{|A \cup B|}$$

No nosso contexto, os conjuntos A e B seriam os conjuntos de notas dos acordes comparados. Uma abordagem comumente adotada na literatura para tratar os casos de acordes mais complexos do dataset é simplificá-los para acordes mais simples, e tratá-los como se o *ground truth* dos mesmos fosse a representação simplificada [1]. Optou-se pela adoção do coeficiente de Jaccard para permitir a flexibilização de nossa análise, caso fosse desejado, sem a necessidade de alterar o *ground truth* dos acordes da base de dados.

Para ilustrar como esse coeficiente é utilizado na comparação de dois acordes, consideremos um exemplo, no qual o acorde esperado é C:9(13) e o acorde previsto foi C:9. Ambos são acordes maiores com extensões, denotados pelos números 9 e 13. O algoritmo previu a tônica (Dó) corretamente, assim como o tipo de acorde, que é maior. No entanto, o algoritmo não detectou que o acorde deveria ter a nota Lá, que nesse caso é denotada pelo número 13. Como ambos acordes são maiores, eles têm as notas da tríade maior, que são a tônica, terça maior e quinta. Além disso, eles têm outra nota em comum que é a nona.

Assim, o coeficiente de Jaccard nesse caso é dado por:

$$J = \frac{|A \cap B|}{|A \cup B|} = \frac{4}{5} = 0.8$$

Em resumo, para considerar que dois acordes são iguais, o algoritmo que realiza a análise de performance dos modelos verifica se:

(1) As tônicas dos acordes foram equivalentes (incluindo casos de notas enarmônicas).
(2) Os tipos dos acordes são iguais.
(3) O coeficiente de Jaccard calculado entre o acorde previsto e o acorde esperado é maior/igual ao threshold definido na análise. Por exemplo, se o threshold fosse igual a 0.8 no exemplo anterior, os acordes seriam considerados iguais.

[3] O repositório contendo os códigos utilizados, assim como o link para o notebook Google Colab, se encontra em: <https://github.com/adrianmartinez-cg/chord_rec_scripts>

4. RESULTADOS E DISCUSSÕES

4.1 Composição do dataset

Como existem 125 tipos de acordes diferentes no dataset, foi necessário agregar as categorias de acordes na hora de visualizar os erros cometidos pelos modelos. Essa agregação foi feita analisando quais são os tipos mais presentes, removendo os tipos mais raros e juntando os mesmos em uma categoria em comum que faça sentido. Com essa simplificação o dataset pode ser representado por 21 categorias de acordes diferentes, e a composição do dataset se mostra da seguinte forma (apenas os 14 mais comumente encontrados estão mostrados na tabela).

Acorde	Porcentagem	Acorde	Porcentagem
maj	41,92%	maj6	1,60%
min	22,11%	min9	1,31%
min7	15,00%	maj9(13)	0,47%
7	5,12%	hdim7	0,33%
maj9	4,37%	maj/b7	0,32%
maj7	4,31%	dim	0,27%
sus4	1,80%	min6	0,25%

Tabela 4 - Tipos de acordes com maior presença no dataset, na representação simplificada

A partir dessa tabela podemos notar que tipos de acordes mais simples são mais predominantes que acordes complexos. Além disso, vemos que não temos tanta variedade de acordes no dataset.

4.2 Avaliação dos modelos de reconhecimento de acorde

4.2.1 Avaliação do modelo BTC-ISMIR19

4.2.1.1 Arquitetura

Este modelo se baseia na utilização de uma rede transformer bi-direcional. A vantagem da utilização dessa rede, é que possui uma velocidade de treinamento consideravelmente mais rápida, enquanto pode alcançar uma performance comparável a abordagens baseadas em redes convolucionais ou recorrentes [7]. A estrutura básica do modelo está mostrada na figura 14. O primeiro bloco consiste na aplicação da transformada CQT (Constant Q-Transform) sobre o sinal de áudio, que é similar a transformada de Fourier, porém tem uma resolução de frequência variável. No segundo bloco, é realizada a operação de dropout, que é um método usado para prevenir o overfitting [22]. Vemos também a presença de camadas completamente conectadas. O bloco com coloração vermelha corresponde à camada que implementa o mecanismo de auto-atenção bidirecional.

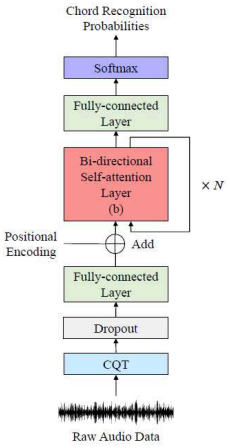

Figura 14 -Arquitetura do modelo BTC-ISMIR19. Fonte: [7]

4.2.1.2 Performance

Threshold para o coeficiente de Jaccard	Precisão	Recall	F1-score
≤ 0.5	80,2%	80.01%	80,07%
0.75	78,71%	78,52%	78,59%
0.8	55,18%	55,07%	55,11%
1	55,00%	54,90%	54,93%

Tabela 5 - Métricas de performance do modelo BTC-ISMIR19

Algo que podemos notar a partir da tabela é que os valores de precisão e recall são bastante próximos, isso se deve à que os tamanhos das linhas do tempo do dataset e as geradas pelo modelo são praticamente iguais. Para valores de threshold abaixo de 0.5, as métricas se mantiveram constantes. Isso se deve a que o menor valor de coeficiente de Jaccard possível na base de dados, considerando que dois acordes tem tipos iguais, é 0.5. Se flexibilizarmos a possibilidade de acordes terem tipos diferentes, de forma a calcular a performance do modelo na previsão da nota tônica do acorde, tivemos um F1-score igual a 83,9%.

Vemos que o modelo teve uma boa performance na inferência da tônica e dos tipos dos acordes. Foi observado uma forte queda nas métricas de performance a partir do valor de threshold aproximadamente igual a 0.8, onde o F1-score já se aproxima bastante do valor visto na última linha da tabela, que corresponde à situação onde os acordes devem possuir exatamente as mesmas notas. Uma forma comum de analisar os erros cometidos por um modelo é utilizar a matriz de confusão. Então, a matriz de confusão para o modelo foi plotada, na situação do threshold máximo para o coeficiente de Jaccard[4]. Alguns dos erros que se destacaram na matriz foram os seguintes:

[4] Os gráficos podem ser visualizados no notebook Google Colab, cujo link está disponibilizado no repositório do trabalho.

Tipo esperado	Tipo inferido	Quantidade
min7	min	15.246 (8.25%)
maj	min	15.171 (8.21%)
min	maj	12.714 (6.88%)
min7	maj	8.408 (4.55%)
min	min7	7.025 (3.80%)
7	maj	6.298 (3.41%)

Tabela 6 - Erros comuns cometidos pelo modelo BTC-ISMIR19, para a situação de rigidez máxima

Vemos que alguns dos erros mais comuns foram a troca de acordes maiores por menores, e vice-versa, e também erros relacionados a notas faltantes, como visto nas linhas 1 e 6 da Tabela 6. Dada a composição do dataset, é de se esperar que a matriz de confusão tenha valores bem maiores para certas combinações de tipos de acordes.

Podemos estar interessados em ter alguma espécie de medida para saber para quais tipos de acordes o modelo acertou ou errou mais, não em quantidade de vezes em relação ao total, mas sim em relação às vezes que o mesmo aparece. Uma forma de obter essa medida é, de forma similar ao que foi descrito na seção 3.2, calcular o quanto os intervalos de tempo dos acordes presentes na linha do tempo descrita pelo dataset, foram cobertos pelo algoritmo de reconhecimento, individualmente para cada acorde, em vez de analisar apenas o total.

Assim , esse cálculo individual foi incorporado no algoritmo que realiza essa análise. Na tabela 7, vemos para quais tipos de acordes o modelo BTC-ISMIR19 apresentou melhor desempenho, com Jaccard igual a 1. Vale ressaltar que cada categoria de acorde da tabela está agregando diversos acordes presentes no dataset.

Acorde	Cobertura	Cobertura / Cobertura Total
maj	75,53%	57,72%
min	72,25%	32,52%
min7	27,05%	6,95%
7	23,54%	2,01%
aug	18,31%	0,02%

Tabela 7 - Tipos de acordes que foram melhor cobertos pelo modelo BTC-ISMIR19, na situação de rigidez máxima.

Observando a tabela 7, vemos que o modelo consegue capturar bem os acordes maiores e menores mais simples, justificando assim as métricas de performance observadas, uma vez que representam boa parte do dataset, enquanto que acordes mais complexos, não foram bem capturados, ao menos considerando a

situação em que são esperados acordes com notas exatamente iguais. Isso pode ser entendido ao observamos que o modelo é limitado em relação aos tipos de acordes utilizados nas predições, onde os tipos estão limitados ao conjunto {maj, min, dim, aug, min6, maj6, min7, minmaj7, maj7, 7, dim7, hdim7, sus2, sus4} [7], enquanto que o dataset possui 125 tipos de acordes diferentes. Diminuindo o threshold do coeficiente de Jaccard para 0.75, e assim flexibilizando um pouco os acertos em relação às extensões dos acordes, obtemos a seguinte tabela.

Acorde	Cobertura	Cobertura / Cobertura Total
min	85,09%	26,51%
maj	81,52%	43,12%
maj7	81,43%	4,32%
min7	80,49%	14,28%
maj9	74,88%	4,64%

Tabela 8 - Tipos de acordes que foram melhor cobertos pelo modelo BTC-ISMIR19, em uma situação de menor rigidez em relação às extensões esperadas.

O algoritmo apresentou agora um desempenho geral muito melhor, considerando as categorias de acordes nas quais anteriormente apresentava baixo desempenho. Isso quer dizer que está cometendo pequenos erros em tais categorias de acordes, provavelmente estimando alguma nota a menos. Isso é de certa forma esperado, uma vez que cada uma dessas categorias também engloba acordes harmonicamente complexos, como foi explicado na seção 4.1. Já em relação aos tipos de acordes com pior desempenho, obtivemos que 9 tipos de acordes não tiveram nenhum acerto, para Jaccard igual a 1, como mostra a tabela 9. Reduzindo para 0,75, temos o resultado mostrado na tabela 10.

Tipos de acordes não cobertos		
maj9	maj9(13)	maj7(13)
7(13)	min(11)	min7(11)
min9	maj(11)	sus2

Tabela 9 - Acordes não cobertos pelo modelo BTC-ISMIR19, na situação de rigidez máxima.

Acorde	Cobertura	Cobertura / Cobertura Total
sus2	0%	0%
maj9(13)	0,01%	0%
sus4	3,21%	0,07%
maj7(13)	6,57%	0,01%
dim	10,72%	0,03%

Tabela 10 - Acordes menos cobertos pelo modelo BTC-ISMIR19, em uma situação de menor rigidez.

4.2.2 Avaliação do modelo Chordino

4.2.2.1 Arquitetura

Possui uma arquitetura baseada na adição de uma etapa durante a extração de características, que é a utilização do algoritmo de mínimos quadrados não negativos (NNLS). Para a aquisição do cromagrama, primeiramente é obtido o espectrograma em log-frequência, depois é feito um pré-processamento e é realizada uma transcrição aproximada das notas usando o algoritmo NNLS. Esta transcrição é então convertida em cromagramas, que são denominados de cromagramas NNLS (*NNLS chroma features*) [21].

4.2.2.2 Performance

Threshold para o coeficiente de Jaccard	Precisão	Recall	F1-score
≤ 0.5	78,86%	78.65%	78,73%
0.75	77,35%	77,14%	77,22%
0.8	54,36%	54,24%	54,28%
1	54,20%	54,09%	54,13%

Tabela 11 - Métricas de performance do modelo Chordino

De forma geral, as métricas de performance se mostraram inferiores em comparação às do modelo anterior, mas o modelo também se mostrou robusto em relação à inferência de tipos de acordes mais básicos. O F1-score desse modelo associado ao acerto das tônicas dos acordes foi igual a 82,37%. Analisando a tabela 12, vemos que os erros mais comuns, apontados pela matriz de confusão, na situação de Jaccard igual a 1, são similares ao do modelo anterior, com algumas diferenças como as apontadas pelas duas últimas linhas da tabela 12. A troca do acorde min7 pelo acorde min continua sendo o erro mais prevalente. O modelo apresentou uma melhor performance nos acordes de tríades menores e maiores em relação ao modelo anterior, na situação apontada na tabela 13, assim como nos acordes do tipo aug, dim e hdim7. No entanto, teve uma performance inferior nos acordes min7 e 7.

Tipo esperado	Tipo inferido	Quantidade
min7	min	15.058 (9.58%)
maj	min	14.971 (9.52%)
min	maj	12.009 (7.64%)
min7	maj	8.109 (5.16%)
7	maj	6.306 (4.01%)
maj9	maj	5.375 (3.42%)

Tabela 12 - Erros comuns cometidos pelo modelo Chordino, para a situação de rigidez máxima

Acorde	Cobertura	Cobertura / Cobertura Total
min	78,69%	35,29%
maj	76,77%	58,53%
aug	38,45%	0,04%
dim	21,17%	0,08%
hdim7	19,12%	0,09%

Tabela 13 - Tipos de acordes que foram melhor cobertos pelo modelo Chordino, na situação de rigidez máxima.

Já na tabela a seguir, que mostra os acordes com melhor desempenho considerando Jaccard igual a 0.75, vemos que os resultados são semelhantes aos obtidos no modelo anterior, com uma performance levemente inferior.

Acorde	Cobertura	Cobertura / Cobertura Total
min	84,97%	26,6%
maj	81,6%	43,43%
maj7	80,05%	4,25%
min7	78,36%	13,99%
maj9	72,61%	4,50%

Tabela 14 - Tipos de acordes que foram melhor cobertos pelo modelo Chordino, na situação de menor rigidez.

Nas tabelas 15 e 16, temos os acordes que foram menos cobertos pelo modelo Chordino, considerando as situações descritas.

Tipos de acordes não cobertos		
sus4	maj9(13)	maj7(13)
7(13)	min(11)	min7(11)
min9	maj(11)	sus2

Tabela 15 - Acordes não cobertos pelo modelo Chordino, na situação de rigidez máxima.

Acorde	Cobertura	Cobertura / Cobertura Total
sus2	0%	0%
sus4	0%	0%
min7(11)	10,53%	0,03%
maj7(13)	12,43%	0,02%
maj9(13)	13,68%	0,09%

Tabela 16 - Acordes menos cobertos pelo modelo Chordino, na situação de menor rigidez.

A diferença da tabela 15 para a tabela 9 é que o tipo de acorde sus4 agora aparece, o que mostra que o Chordino não oferece suporte para acordes do tipo suspenso (denotados por sus). Além disso, o tipo maj9, que aparecia na tabela 9, não está na tabela 15.

4.2.3 Avaliação do modelo Autochord

4.2.3.1 Arquitetura

Baseia-se na utilização do plugin VAMP NNLS-Chroma para extração do cromagrama a partir do áudio, e então os cromagramas servem de entrada para um modelo baseado na utilização de uma rede neural recorrente bidirecional com memória de longo prazo (Bi-LSTM) juntamente com o CRF (Conditional Random Field) , que é um modelo estatístico, onde no contexto de reconhecimento de acordes, é utilizado para modelar as relações entre os acordes adjacentes em uma sequência de forma probabilística. Esse modelo é obtido através do uso da biblioteca Tensorflow.

4.2.3.2 Performance

Threshold para o coeficiente de Jaccard	Precisão	Recall	F1-score
≤ 0.5	72,52%	72,33%	72,40%
0.75	71,64%	71,46%	71,52%
0.8	50,44%	50,35%	50,38%
1	50,44%	50,35%	50,38%

Tabela 17 - Métricas de performance do modelo Autochord

Esse modelo se mostrou com as piores métricas de performance dentre os modelos avaliados. O F1-score desse modelo associado ao acerto das tônicas dos acordes foi igual a 75,97%. A tabela 18 mostra os erros mais comuns presentes na matriz de confusão para o modelo na situação mencionada.

Tipo esperado	Tipo inferido	Quantidade
min7	min	18.112 (11.98%)
maj	min	15.966 (10.56%)
min	maj	14.633 (9.68%)
min7	maj	10.829 (7.16%)
7	maj	7.113 (4.70%)
maj7	maj	6059 (4.01%)

Tabela 18 - Erros comuns cometidos pelo modelo Autochord, para a situação de rigidez máxima

Como mostra a tabela 19, os únicos tipos de acordes que foram cobertos pelo modelo, quando se esperam notas exatamente iguais, foram as tríades menores e maiores.

Acorde	Cobertura	Cobertura / Cobertura Total
min	76,12%	37,54%
maj	74,63%	62,46%

Tabela 19 - Tipos de acordes que foram cobertos pelo modelo Autochord, na situação de rigidez máxima.

Na tabela 20, temos os tipos de acordes que foram melhor inferidos pelo modelo, considerando uma maior flexibilização das notas esperadas, considerando o limiar do coeficiente de similaridade igual a 0.75. Já na tabela 21, temos os tipos de acordes que tiveram as métricas de performance zeradas, mesmo considerando essa flexibilização. Assim como o modelo Chordino, acordes do tipo suspenso, não estão sendo tratados. Na tabela 21, também aparecem alguns novos tipos de acordes que tiveram as métricas de performance zeradas, como os acordes do tipo dim, hdim7 e aug, o que não aconteceu em outros modelos analisados até agora.

Acorde	Cobertura	Cobertura / Cobertura Total
maj7	79,03%	4,59%
min	76,15%	26,12%
maj	74,63%	43,44%
maj9	74,51%	5,05%
min7	71,18%	13,91%

Tabela 20 - Tipos de acordes que foram melhor cobertos pelo modelo Autochord, na situação de menor rigidez.

Tipos de acordes não cobertos		
sus2	dim	maj7(13)
aug	hdim7	maj9(13)
7(13)	min7(11)	sus4

Tabela 21 - Acordes não cobertos pelo modelo Autochord, na situação de menor rigidez.

4.2.4 Avaliação do modelo Omnizart

4.2.4.1 Arquitetura

Se baseia na utilização de um componente chamado Harmony Transformer (HT), que é um modelo de aprendizagem profunda para análise de harmonia, baseado em uma arquitetura codificador-decodificador [23]. Especificamente, o codificador realiza a segmentação de acordes na entrada e, posteriormente, o decodificador reconhece a progressão de acordes com base no resultado da segmentação.

4.2.4.2 Performance

Threshold para o coeficiente de Jaccard	Precisão	Recall	F1-score
≤ 0.5	78,68%	75,91%	77,24%
0.75	77,74%	75%	76,31%
0.8	54,52%	52,63%	53,54%
1	54,52%	52,63%	53,54%

Tabela 22 - Métricas de performance do modelo Omnizart

Em relação às métricas de performance, este modelo se mostrou superior ao modelo anterior. O F1-score desse modelo associado ao acerto das tônicas dos acordes foi igual a 81,13%. A tabela 23 mostra os erros mais comuns presentes na matriz de confusão para o modelo para o limiar de coeficiente de similaridade igual a 1. Assim como aconteceu com o modelo Autochord, os únicos tipos

de acorde que foram cobertos nesta situação foram as tríades maiores e menores, dessa vez com a tríade maior apresentando um melhor desempenho, como mostra a tabela 24.

Tipo esperado	Tipo inferido	Quantidade
min	maj	18.817 (11.79%)
min7	min	17.387 (10.90%)
maj	min	14.329 (8.98%)
min7	maj	12.719 (7.97%)
7	maj	7.659 (4.80%)
maj9	maj	6.790 (4.26%)

Tabela 23 - Erros comuns cometidos pelo modelo Omnizart, para a situação de rigidez máxima

Acorde	Cobertura	Cobertura / Cobertura Total
maj	80,99%	62,77%
min	80,40%	37,23%

Tabela 24 - Tipos de acordes que foram cobertos pelo modelo Omnizart, na situação de rigidez máxima.

Nas tabelas a seguir, temos os tipos de acorde que apresentaram melhor e pior desempenho, para a situação na qual o limiar para o coeficiente de similaridade foi igual a 0.75.

Acorde	Cobertura	Cobertura / Cobertura Total
maj7	81,51%	4,44%
maj	80,99%	43,82%
min	80,42%	26%
maj9	77,06%	4,85%
min7	75,85%	13,78%

Tabela 25 - Tipos de acordes que foram melhor cobertos pelo modelo Omnizart, na situação de menor rigidez.

Tipos de acordes não cobertos		
sus2	dim	maj7(13)
aug	hdim7	maj9(13)
7(13)	min7(11)	sus4

Tabela 26 - Acordes não cobertos pelo modelo Omnizart, em uma situação de menor rigidez.

Como é possível ver na tabela 26 , os tipos de acorde que tiveram os piores desempenhos são iguais aos do modelo anterior.

4.2.5 Comparativo entre os modelos

O modelo que apresentou melhores métricas de desempenho no geral foi o BTC-ISMIR19, ainda que o modelo Chordino tenha conseguido melhor desempenho em alguns tipos de acordes como discutido na seção 4.2.2.2. Por fim, os modelos Omnizart e Autochord apresentaram desempenho inferior, principalmente devido ao fato de terem um vocabulário de acordes menor em relação aos modelos anteriores. Analisando de forma geral, as categorias de acordes do dataset que não foram previstos de forma exata por nenhum modelo estão mostrados na tabela a seguir.

Categoria de acorde	Tipos de acordes correspondentes no dataset
sus2	sus2,sus2(4), sus2(6), sus2/5
min(11)	min(11), min/4, min11, min11/5, min11/b3, min11/b7
min7(11)	min7(11), min7(4)/4, min7(4)/b7, min7/4
maj9(13)	maj9(13), maj(9)/6, maj6(2)/2, maj6(9), maj6(9)/3, maj6(9)/5, maj6/2,
maj7(13)	maj7(13)
maj(11)	maj(11), maj(4)/4, maj/4
min9	min9, min(9)/5, min(9)/b3, min9/5, min9/b3, min9/b7, min(2)/2, min/2, min6/2
7(13)	7/6, maj6(b7)

Tabela 27 - Tipos de acordes não cobertos pelos modelos avaliados, para o limiar de coeficiente de similaridade igual a 1

5. CONCLUSÃO

Ao analisarmos modelos de reconhecimento de acordes disponibilizados publicamente, é possível notar que mesmo os classificadores mais precisos ainda não preveem com exatidão as notas de acordes mais complexos harmonicamente, tais como acordes com mais de uma extensão, ou com inversões, principalmente pelo fato de terem o vocabulário limitado. Além disso, a concepção de um modelo que seria capaz de fazer isso, certamente é desafiador, dada a enorme variedade de acordes existentes, como mostra a tabela 27, e assim a construção de um dataset adequado para realizar o treinamento seria custoso, principalmente em relação à construção do *ground truth,* que exigiria especialistas com alto grau de expertise no assunto. Apesar das limitações apontadas, os erros cometidos pelos melhores modelos normalmente são pequenos, e assim podem ser aplicados de maneira satisfatória em contextos mais simples. Dentre os modelos avaliados, o que se mostrou com performance mais promissora é o modelo baseado em arquitetura transformers.

6. REFERÊNCIAS

[1] HAUSNER, C. Design and Evaluation of a Simple Chord Detection Algorithm. University of Passau - Faculty of Computer Science and Mathematics, 2014.

[2] BORTOLOZZO, M. Improving Rare Chord Recognition Through Self-Learning Techniques and Weak Label Generation. Universidade Federal do Rio Grande do Sul, 2022.

[3] WANG, Ziyu et al. Pop909: A pop-song dataset for music arrangement generation. arXiv:2008.07142, 2020.

[4] LENSSEN, N. Applications of Fourier Analysis to Audio Signal Processing: An Investigation of Chord Detection Algorithms. Senior Thesis. Claremont McKenna College, 2013.

[5] GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. 4. ed. New York, Ny: Pearson, 2018.

[6] An Introduction To Pitch, Pitch Classes, And Octaves. Disponível em: <https://musicalsanctuary.com/pitch-classes-octaves>. Acesso em: 21 abr. 2024.

[7] PARK, J. et al. A Bi-Directional Transformer for Musical Chord Recognition. 20th International Society For Music Information Retrieval Conference, 2019.

[8] KORZENIOWSKI, F.; WIDMER, G. Feature Learning for Chord Recognition: The Deep Chroma Extractor. 17th International Society for Music Information Retrieval Conference, 2016.

[9] CHO, T.; BELLO, J. P. On the Relative Importance of Individual Components of Chord Recognition Systems. IEEE/ACM Transactions on Audio, Speech, and Language Processing, v. 22, n. 2, p. 477–492, fev. 2014.

[10] T. Fujishima. Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music. Proceedings of the International Computer Music Conference (ICMC), Beijing, China, 1999.

[11] PETRUSHIN, V. Hidden Markov Models: Fundamentals and Applications Part 1: Markov Chains and Mixture Models. Disponível em: <https://www.eecis.udel.edu/~lliao/cis841s06/hmmtutorialpart1.pdf> Acesso em: 1 maio. 2024.

[12] MALTBY, H.; PAKORNRAT, W.; JACKSON, J. Markov Chains | Brilliant Math & Science Wiki. Disponível em: <https://brilliant.org/wiki/markov-chains/>. Acesso em: 1 maio. 2024.

[13] Hidden Markov Models for Chord Recognition, Intuition and Applications - Caio Miyashiro - PyConDE & PyDataBerlin 2019 conference. Disponível em: <https://2019.pycon.de/program/pydata-yxndb9-hidden-markov-models-for-chord-recognition-intuition-and-applications-caio-miyashiro/>. Acesso em: 1 maio, 2024.

[14] LÓPEZ, N. Automatic Roman Numeral Analysis in Symbolic Music Representations. McGill University - Department of Music Research, 2022.

[15] HARDESTY, L. Neural networks explained. Disponível em: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>. Acesso em: 3 maio, 2024.

[16] ROBERTS, Daniel A.; YAIDA, Sho; HANIN, Boris. The principles of deep learning theory. Cambridge University Press, 2022.

[17] IBM. What is Deep Learning? Disponível em: <https://www.ibm.com/topics/deep-learning>. Acesso em: 5 maio, 2024.

[18] What deep learning is and isn't. Disponível em: <https://thedatascientist.com/what-deep-learning-is-and-isnt/>. Acesso em: 5 maio, 2024.

[19] VASWANI, Ashish et al. Attention is all you need. Advances in neural information processing systems, v. 30, 2017.

[20] Convolutional Neural Networks. Disponível em: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-understand-the-basics/>. Acesso em: 5 maio, 2024.

[21] MAUCH, Matthias; DIXON, Simon. Approximate Note Transcription for the Improved Identification of Difficult Chords. In: ISMIR. 2010. p. 135-140.

[22] SRIVASTAVA, Nitish et al. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, v. 15, n. 1, p. 1929-1958, 2014.

[23] WU, Yu-Te et al. Omnizart: A general toolbox for automatic music transcription. arXiv:2106.00497, 2021.