UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

MARCELO BROSOWICZ DE PAULO

GERAÇÃO DE MÚSICA COM MACHINE LEARNING

FLORIANÓPOLIS
2022

Marcelo Brosowicz de Paulo

Geração de Música com Machine Learning

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Sistemas de Informação
Orientador: Prof. Elder Rizzon Santos, Dr.

Florianópolis

2022

Marcelo Brosowicz de Paulo

Geração de Música com Machine Learning

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em curso de Sistemas de Informação.

Florianópolis, 3 de Agosto de 2022.

_____

Prof. Álvaro Junio Pereira Franco, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Elder Rizzon Santos, Dr.
Orientador
Instituição UFSC

_____

Prof. Guilherme Alex Derenievicz, Dr.
Instituição UFPR

_____

Prof. Rafael de Santiago, Dr.
Instituição UFSC

AGRADECIMENTOS

O trabalho aqui apresentado foi nada menos que um grande desafio, tanto pessoal quanto profissional. Precisei de muita força, vontade e acima de tudo, superação para finalizá-lo e deixá-lo em um estado do qual eu sentisse orgulho e um sentimento de dever cumprido. Nada disso seria possível sem pessoas importantes que me ajudaram de diversas formas ao longo do processo, das quais sou eternamente grato.

Um obrigado especial aos meus pais, Gilnei Pereira de Paulo e Lisiane Brosowicz. São as pessoas que fazem de tudo por mim independente da situação, sempre com serenidade e amor. Vocês são parte fundamental do que foi desenvolvido aqui, afinal, sem vocês esse projeto nunca existiria. Quero que saibam o carinho eterno que tenho por ambos.

Agradeço também ao meu orientador, o prof. Elder Rizzon Santos, que desde o início acreditou na minha ideia e me fez ver uma luz em momentos de confusão ou dúvidas em relação ao projeto. Foi um prazer trabalhar ao seu lado, completando um marco tão importante com um professor excepcional, que faz a sua ser uma das aulas mais interessantes de todo currículo do curso, não à toa sendo querido e reconhecido por inúmeros alunos.

Por fim, mas não menos importante, obrigado aos meus amigos por me darem ouvidos para reclamar, ombros para chorar e me fazerem dar risadas quando foi necessário. Vocês são todos incríveis.

They gon' try and tell you no, shatter all your dreams.
But you gotta get up, go to bigger, better things.

Mac Miller, Live Free

RESUMO

A inteligência artificial e o aprendizado de máquina são técnicas computacionais muito relevantes no momento, o que torna seu uso no campo criativo cada vez mais comum. A composição e produção musical são processos em grande parte realizados por humanos e facilitados por ferramentas computacionais, mas imensamente dependentes da compreensão humana de áreas emocionais e criativas envolvidas no produto final e o que ele pode proporcionar ao ouvinte. Hoje em dia, diversos experimentos buscam compreender e aplicar métodos geracionais para criar faixas que se assemelham àquelas criadas por seres-humanos ou, pelo menos, sonoramente agradáveis. O principal objetivo deste trabalho consiste na pesquisa e análise da produção musical a partir de entradas do usuário (como melodias, músicas) utilizando técnicas de aprendizado de máquina, com intuito de propor um protótipo de aplicação capaz de gerar melodias com notas que acompanhem uma batida de bateria fornecida ao programa pelo usuário em formato MIDI. Para isto, o estado da arte em aprendizado de máquina para geração musical é analisado de acordo com a extensa busca por subsídios relevantes no meio científico desta área de concentração, selecionando o modelo mais adequado para o treinamento no contexto específico apresentado e analisando a aplicabilidade de diferentes tipos de aprendizagem, como reforço, não supervisionado e supervisionado. Experimentos são realizados para geração de melodias de piano e os resultados de ditos experimentos são também devidamente analisados e comparados com experimentos semelhantes.

Palavras-chave: Aprendizado de máquina. Geração Musical. Modelos.

ABSTRACT

Artificial intelligence and machine learning are currently very relevant computational techniques, which makes their use in the creative field increasingly common. Music composition and production are processes largely performed by humans and facilitated by computational tools, but immensely dependent on human understanding of emotional and creative areas involved in the final product and what it can deliver to the listener. Nowadays, several experiments seek to understand and apply generational methods to create tracks that resemble those created by humans, or at least sound pleasant. The main goal of this work is to research and analyze the musical production from user input (such as melodies, songs) using machine learning techniques, in order to propose a prototype application capable of generating melodies with notes that accompany a drum beat supplied to the program by the user in MIDI format. For this, the state of the art in machine learning for music generation is analyzed according to the extensive search for relevant subsidies in the scientific environment of this area of concentration, selecting the most appropriate model for training in the specific context presented and analyzing the applicability of different types of learning, such as reinforcement, unsupervised and supervised. Experiments are conducted for generating piano melodies and the results of said experiments are also duly analyzed and compared with similar experiments.

Keywords: Machine learning. Musical Generation. Models.

SUMÁRIO

1 INTRODUÇÃO

A música é uma linguagem universal, conforme concluído por Kashyap (2021). Seres humanos a utilizam desde a era paleolítica para se comunicarem e expressarem sentimentos entre si. Ao longo da história, compositores surpreendentes e cativantes surgiram sem aparentes padrões e razões claras para seus sucessos, trazendo consigo muitas vezes estilos, conceitos e técnicas nunca vistas e praticamente impossíveis de serem previstas. Músicos estes que revolucionaram o processo criativo com novos estilos musicais compondo peças inovadoras e/ou muito precisos na criação de peças com uma abundância de estrutura musical subjacente. É possível então que o computador também consiga aprender a criar essa estrutura musical?

O aprendizado de máquina (ML) está relacionado a um conceito muito anterior chamado inteligência artificial (IA) — consolidado por volta da década de 50 — com redes neurais simples e modelos ainda muito novos, que vieram evoluindo exponencialmente com a história e popularizando-se cada vez mais como um importante meio de fornecer aos sistemas a autonomia, ou seja, a capacidade de aprender e melhorar automaticamente a partir da experiência sem ser explicitamente programado, como visto na abordagem do estudo por Ongsulee (2017). O aprendizado de máquina centra-se no desenvolvimento de “softwares” que podem acessar dados e usá-los para aprender por si mesmos.

Recentemente, houve grande progresso na aplicação do aprendizado de máquina a uma gama de problemas relacionados à música, como "*thumb-nailing*" (HUANG, CHOU e YANG, 2018), geração de música (ROBERTS et al., 2018) e transferência de estilo (LU, 2019). Para demonstrar o resultado desses modelos de aprendizagem de máquina no contexto criativo e musical, os pesquisadores geralmente colocam a saída de áudio como o resultado nos sites dos projetos.

Apesar do processo realizado e criatividade ser um dos assuntos mais populares em inteligência artificial, especialistas se perguntam, no entanto, até onde a IA pode ou deve ir no processo criativo. A IA já ajudou a escrever baladas pop, imitou os estilos de grandes pintores e informou decisões criativas na produção de filmes, mas quão presente no contexto criativo ela deve ou pode estar? Um dos problemas na aplicação do aprendizado de máquina e IA no processo de criação musical é o quão "humanizadas" e bem colocadas serão as saídas, que será abordado neste trabalho.

1.1 DESCRIÇÃO DO PROBLEMA

Sistemas que aproveitam do aprendizado de máquina para garantir certa autonomia ao computador geralmente dependem de níveis abundantes de dados para garantirem uma boa margem de acerto em suas saídas, bem como dependem de tempo de treinamento e processamento para acelerar este processo, conforme comentado por Brownlee (2017). Porém, não há valor exato de qual o nível de densidade de dados é necessário, pois isso depende da complexidade do problema e da complexidade do algoritmo de aprendizado.

É possível aplicar o conceito de analogia, logo que diversas pessoas já trabalharam em muitos problemas de aprendizado de máquina aplicados e algumas delas acabam publicando seus resultados. Ademais, a estrutura de longo prazo é um conceito interessante que pode se tornar um desafio considerável nesta categoria de estudo, buscando gerar composições de minutos de duração com estrutura convincente. Funções de humanização têm sido incorporadas em softwares de produção de música conhecidos da indústria (como o Ableton) por diversos anos, mas a evidência aponta que os métodos utilizados atualmente e são aplicados nessas categorias de kits de ferramentas profissionais (com timings de notas aleatórios e dinâmicas com ruído gaussiano, por exemplo) não costumam surtir efeito no usuário final, ou seja, no ouvinte, como visto no estudo de Senn (2018).

Neste trabalho, em vez de gerar novos conteúdos tendenciosos e direcionados a um gênero fixo como o rap ou rock — que podem tornar a aplicação nichada, sua aplicabilidade limitada e gerar resultados que podem pender ao plágio ou à monotonia — o interesse vem com intuito de aprender a traduzir representações de ideias que são mais facilmente expressas (abstrações musicais como arquivos MIDI) para instâncias que tornem o conteúdo realista, de uma maneira agradável e que dá vida a uma ideia, encontrando equilíbrio com referências reais em um ambiente controlado e que de outra forma só seriam produzidas por aqueles qualificados num determinado instrumento (performances), similar ao que ocorre conforme o estudo sobre performances expressivas de bateria geradas pelo computador de Gillick et al. (2019). Assim sendo, o uso de ML em contextos como este busca entender as relações intrínsecas, teóricas e até humanas, da composição musical automaticamente, podendo gerar acompanhamento musical convenientemente para aquele que busca matéria-prima para um arranjo simples ou complexo. Essa experimentação comumente utiliza arquivos MIDI como categorias de valores de entrada, cada um se adequando a diferentes modelos, técnicas ou métodos de aprendizado de máquina. Atualmente, apesar do contrário ser possível, não existe uma maneira de gerar melodias de piano para acompanhamento de batidas de bateria com as ferramentas descritas e utilizadas neste trabalho, assim, o projeto aqui descrito visa resolver este problema.

1.2 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos deste TCC.

1.2.1 *Objetivo Geral*

O principal objetivo deste trabalho consiste na produção musical através de um protótipo capaz de gerar melodias com notas que acompanham um conjunto de entradas do usuário (sendo sempre faixas MIDI com batidas de bateria) utilizando técnicas de Machine Learning.

1.2.2 *Objetivos específicos*

Nesta seção, estão dispostos em listagem a descrição dos objetivos específicos deste estudo, com subitens expandindo brevemente o que é esperado de cada item definido anteriormente. A descrição do objetivo busca apresentar a ideia geral do que está sendo proposto naquele determinado item, enquanto as ações nos subitens demonstram o que é feito para alcançar tais objetivos.

- Analisar o estado da arte no processo de construção musical, incluindo notas, escalas, campos harmônicos, arranjos, ritmos, batidas, melodias, etc.
    - Estudar e compilar informações de artigos, monografias e dissertações prévias relacionados à pesquisa na área de produção musical visando concluir o nível mais alto de desenvolvimento desta arte
- Analisar o estado da arte em Machine Learning para geração musical
    - Estudar, compilar informações e realizar comparativos com artigos, monografias e dissertações prévias relacionados à pesquisa na área de Machine Learning para geração musical visando concluir o nível mais alto de desenvolvimento desta arte
- Selecionar modelos mais adequados para treinamento visando gerar saídas com melodias baseadas em entradas do usuário
    - Baseando-se na consonância e em desempenho, treinar e testar diferentes modelos de aprendizado para então comparar e selecionar o mais capaz de gerar melodias condizentes com as entradas fornecidas pelo usuário, considerando principalmente métricas baseadas em altura e ritmo
- Realizar experimentos para geração de melodias

- Desenvolver um protótipo de plugin capaz de gerar melodias com notas que acompanham uma entrada do usuário na categoria de batida, utilizando o(s) modelo(s) selecionado(s)

- Analisar os resultados de ditos experimentos
  - Revisar o funcionamento do plugin e os resultados finais da geração. Neste caso, considerando o desempenho do plugin e dos modelos utilizados, e a consonância das melodias geradas em relação às entradas do usuário. Segundo Terhardt (1984), a consonância é um dos princípios responsáveis por definir uma série de sonoridades como estáveis e agradáveis aos ouvidos, com base especialmente nos intervalos, timbre e altura.

1.3 JUSTIFICATIVA

A chegada de métodos de aprendizado de máquina no âmbito de modelagem generativa potencializaram a criação de importantes e convincentes modelos. Avanços recentes mostram, por exemplo, que é possível criar legendas a partir de imagens, como visto nos trabalhos de Pu et al. (2016) e Vinyals et al. (2016); imagens de legendas, apresentado nos estudos de Mansimov et al. (2015) e Reed et al. (2016); e até áudio a partir de texto, explicitado nos textos de Gibiansky et al. (2017) e Oord et al. (2016). Independente disso, muitas dessas técnicas dificilmente são aplicáveis para o domínio da geração de música, visto que a música tem estrutura e regras próprias e é intrinsecamente diferente de qualquer outro tipo de dado. A elaboração de modelos generativos para música cria inúmeros novos e interessantes desafios computacionais.

Uma pergunta natural que é feita frequentemente quando se fala sobre a geração automática de música é a seguinte: Para quê? Por que abriríamos mão de nossas faculdades artísticas aos computadores e nos tornaríamos meros operadores? (HADJERES, 2018)

A ideia por trás de propor um protótipo de aplicação capaz de gerar melodias de piano com notas que complementam uma batida de bateria é de tentar contribuir para a evolução desta emergente e questionada tecnologia em um dos campos mais experimentais para os sistemas computacionais inteligentes, a criatividade. Com as análises feitas, o protótipo pode potencializar a criatividade ou a produtividade dos usuários, visando uma frutífera interação homem-máquina em um escopo limitado, que também pode ajudar a ter uma melhor compreensão de como modelos generativos gerais e musicais se comportam.

1.4 MÉTODO DE PESQUISA

Este trabalho é classificado como uma pesquisa aplicada, o método científico específico que envolve a aplicação prática da ciência sendo útil para encontrar soluções para problemas cotidianos, geralmente direcionado para a um problema prático (FONTELLES, 2009) e que, no momento de escrita, abrange o conhecimento necessário para o treinamento de modelos de aprendizado de máquina — analisando o conceito e aplicando práticas da aprendizagem por reforço — e desenvolvimento de um protótipo para avaliação de sua aplicabilidade no ramo criativo da criação musical.

Esta pesquisa tem uma abordagem quantitativa, onde são utilizadas diferentes técnicas estatísticas para quantificar resultados dos experimentos e informações para este determinado estudo. Ela é realizada para compreender e enfatizar o raciocínio lógico e todas as informações que se possam mensurar sobre as experiências humanas. (FONTELLES, 2009).

Por fim, direcionando aos objetivos, esta pesquisa é considerada exploratória, que "visa a uma primeira aproximação do pesquisador com o tema, para torná-lo mais familiarizado com os fatos e fenômenos relacionados ao problema a ser estudado" (FONTELLES, 2009, *p. 6*). Em virtude deste contexto, se mostra necessária uma busca extensiva de subsídios para definição e entendimento das relações presentes neste estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos envolvidos no trabalho desenvolvido para melhor compreensão.

2.1 APRENDIZADO DE MÁQUINA

Para Michie et al. (1994), o aprendizado de máquina — ou Machine Learning — pode ser considerado como computadores capazes de aprender com a experiência, visando melhorar automaticamente a eficiência de seus próprios programas durante a execução. Já Faul (2019) prefere primeiro definir aprendizado como um fenômeno plurifacetado, que consiste de processos como a descoberta de novos fatos e teorias através da observação e experimentação. Então, a pesquisa e digitalização de processos de aprendizagem em diferentes níveis e representações constituem o aprendizado de máquina.

2.1.1 Aprendizado por reforço

Aprendizado por reforço é uma categoria do aprendizado de máquina, em que o agente aprende a atingir uma meta em um ambiente incerto e potencialmente complexo. Sua promessa é uma maneira de programar agentes de recompensa e punição, sem precisar especificar como a tarefa deve ser realizada. (KAELBLING, 1996)

Ao longo dos tempos, diversos algoritmos de aprendizado por reforço foram desenvolvidos. A decisão de qual algoritmo de aprendizado em particular deve ser utilizado é um passo essencial. Segundo Szepesvári (2010), algoritmos de aprendizado por reforço podem ser pensados como uma forma de transformar métodos inviáveis de programação dinâmica em algoritmos práticos, para que possam ser aplicados a problemas em larga escala.

Há duas tarefas fundamentais de aprendizado de reforço: previsão e controle. Nas tarefas de previsão, nos é dada uma *policy* e nosso objetivo é avaliá-la através da estimativa do valor ou valor Q de tomar ações seguindo esta *policy*. Nas tarefas de controle, não conhecemos a *policy*, e o objetivo é encontrar a *policy* ideal que nos permita coletar a maioria das recompensas. (XU, 2020)

2.1.1.1 Actor-Critic

Os problemas relacionados ao aprendizado por reforço normalmente se encaixam em uma das seguintes categorias: Métodos *actor-only* que trabalham com uma família parametrizada de *policies;* ou métodos *critic-only* que se baseiam exclusivamente na aproximação da função de valor e buscam aprender uma solução aproximada para a equação de Bellman, que então será utilizada para prescrever uma *policy* otimizada, segundo Konda e Tsitsiklis (2000).

Métodos *actor-critic* visam mesclar os métodos, explorando os pontos fortes de ambos. A parte crítica utiliza uma arquitetura de aproximação e simulação para aprender uma função de valor, que é então utilizada para atualizar os parâmetros da *policy* do ator buscando melhoria de desempenho. (KONDA e TSITSIKLIS, 2000)

O estudo de Bahdanau et al. (2016) explora o contexto da abordagem *actor-critic* para geração de sequências, uma abordagem particularmente interessante para geração musical. Foram utilizadas redes adicionais e aplicadas tarefas que podem ser aprendizagem supervisionada, permitindo acesso a valores de referência eventualmente para uso como entrada pela parte crítica. O fundamento teórico deste método é que, sob a suposição de que a parte crítica calcula valores exatos, a expressão usada para treinar o ator é uma estimativa imparcial do gradiente da pontuação específica da tarefa esperada. (BAHDANAU et al., 2016)

Jiang et al. (2020) aplicaram a abordagem *actor-critic* em conjunto com *generalized advantage estimator* (GAE), visto em estudos como o de Schulman et al. (2016), para treinar o agente de geração do seu modelo, que é capaz de colaborar com humanos para criar música e acompanhamento. Segundo Jiang et al. (2020), o *actor-critic* com GAE aprende uma política de ação e uma função de valor e, dessa forma, o agente dá uma ação com base no estado atual, otimizando a recompensa total esperada a longo prazo.

2.1.2 Aprendizado não supervisionado

Aprendizado não supervisionado é outra categoria do aprendizado de máquina, em que o conjunto de dados não apresenta nenhuma categoria de rótulo. Dessa forma, considerando os objetivos finais da máquina (tomada de decisões, previsão de entradas futuras, comunicação eficiente das entradas para outra máquina, etc.), é possível construir uma uma estrutura formal de aprendizagem não supervisionada, segundo Ghahramani (2003). Em certo sentido, a aprendizagem não supervisionada pode ser pensada como encontrando padrões nos dados acima e além do que seria considerado puro ruído não estruturado. (GHAHRAMANI, 2003)

De acordo com Celebi e Aydin (2016), as principais e mais populares subcategorias do aprendizado não supervisionado são o *clustering* e a detecção de anomalias. Anomalia pode ser amplamente definida como algo que se afasta do que geralmente é considerado normal (ou seja, comum) (CELEBI E AYDIN, 2016). É um conceito similar ao de *outlier*, mas menos granular e individual, visto que um grupo de objetos pode ser considerado como uma anomalia se seu comportamento coletivo se desviar do que é considerado comum, segundo Celebi e Aydin (2016). *Clustering* é uma tarefa de classificação, cujo objetivo é separar um conjunto de dados finito e não rotulado (no caso do aprendizado não supervisionado), em um conjunto finito e discreto de padrões e estruturas "naturais" escodidas nos dados, segundo Dayan e Sahani (1999).

2.1.2.1 Variational Autoencoder (VAE)

Antes de definir o autocodificador variável, ou *Variational Autoencoder* (VAE), é necessário entender o que é o *Autoencoder* (AE). AE é uma rede neural que aprende como codificar a entrada em dimensões mais baixas, depois decodificar e reconstruir os dados para estar o mais próximo possível da entrada, da maneira mais eficiente possível, como descrito por Rashad (2020). Ele é consistido pelo codificador, camadas que codificam os dados de entrada em uma representação de dimensões mais baixas; compressor (ou *Bottleneck*), camada que contém a representação codificada/comprimida e a dimensão mais baixa; e pelo descodificador, camadas que aprendem a decodificar ou reconstruir a interpretação codificada para os dados tão próximos quanto os de entrada.

Sobre os resultados do AE, Rashad coloca:

> Os vetores latentes (codificação) gerados pelo codificador tendem a ser irregulares, desorganizados ou não interpretáveis, já que ele só visa reconstruir a entrada da maneira mais semelhante possível sem qualquer restrição do espaço latente. Portanto, não importa como ele codifica os dados desde que possa reconstruir a entrada perfeitamente. (RASHAD, 2020)

O VAE é a instância mais popular do algoritmo *Auto-encoding variational Bayes* (AEVB), proposto por Kingma e Welling (2013). Ele é, na verdade, um modelo generativo. Um modelo generativo simula como os dados são gerados no mundo real (KINGMA e WELLING, 2019). Ele utiliza o algoritmo AEVB para aprender um modelo específico usando um determinado codificador e foi profundamente revisitado pelos autores Kingma e Welling (2019). Ademais, a geração considera a regularização do espaço latente como sendo contínuo, o que permite uma interpolação suave entre diferentes atributos e remove lacunas onde possam ser retornadas saídas não-realistas, também comentado por Rashad (2020). Desde sua primeira idealização, já teve seu potencial e aplicação amplamente reconhecidos, sendo utilizado em estudos para: gerar frases a partir de um espaço contínuo, como o de Bowman et al. (2015); gerar de imagens, como visto no estudo de Vahdat e Kautz (2020); constituir um sistema de recomendação, no estudo de Li e She (2017) e geração musical, chegando a contextos ainda mais específicos como geração de padrões de bateria, como descrito por Wei, Wu e Su (2019).

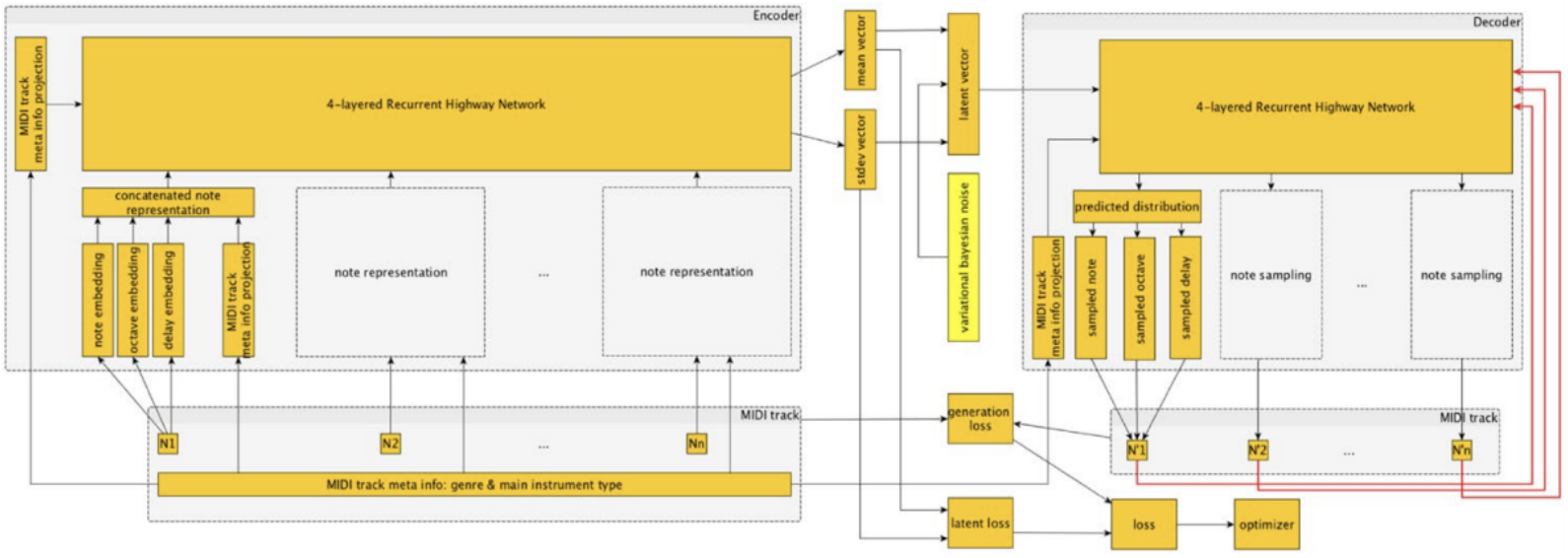


Figura 1 — Variational autoencoder recorrente apoiado por esquema histórico (VRASH) para geração de música

(YAMSHCHIKOV e TIKHONOV, 2020)

Para geração musical, tem sido particularmente eficiente. A estrutura planejada pelos seus criadores originais — que pode ser observada na Figura 1 — se adequa ao problema perfeitamente, dado que o VRASH proporciona um bom equilíbrio entre a estrutura global e a local da faixa, além de gerar padrões diversificados, ser relativamente fácil de implementar e de treinar, e permitir o controle sobre o estilo da saída. Assim, o VRASH obtém exemplos semelhantes aos que são extraídos da distribuição dos dados de entrada, além de controle significativo sobre os parâmetros da saída, segundo Yamshchikov e Tikhonov (2020). Basicamente, ele acrescenta um ponto de vista probabilístico à arquitetura do autocodificador, introduzindo a possibilidade de aprender a distribuição de dados e de amostrar a partir dela, segundo Hadjeres (2018). Neste estudo, o *variational autoencoder* é combinado com uma nova arquitetura de rede neural artificial e com heurísticas de filtragem, o que permite a geração de música acusticamente agradável e melodicamente diversificada, segundo Yamshchikov e Tikhonov (2020).

2.2 MINERAÇÃO DE DADOS

Mineração de dados e aprendizado de máquina são campos de estudo muitas vezes confundidos. A mineração de dados é o processo de identificação de padrões interessantes a partir de grandes bancos de dados, sendo a parte central do processo de descoberta de conhecimento em banco de dados (KDD). A mineração de dados e o KDD são frequentemente usados de forma intercambiável porque a mineração de dados é a chave para o processo KDD. (FU, 1997)

Mineração de dados abrange uma vasta gama de tarefas, visto que bancos de dados de tamanho considerável possuem inúmeros padrões. Para encontrar todos os tipos de padrões, se fazem necessários diferentes métodos e técnicas. Segundo Fu (1997), esses tipos são: *summarization; classification; clustering; association* e *trend analysis.*

Para Aggarwal (2015), um aspecto extremamente relevante é como os dados brutos podem ser arbitrários, não estruturados, ou estar em um formato que não seja de pronto apropriado para o processamento automatizado. Por exemplo, os dados podem ter sido coletados manualmente e extraídos de diferentes fontes em formatos variados e, de alguma maneira, precisam ser processados por um programa automatizado visando gerar novas intuições a partir de conhecimentos e informações.

2.3 MINERAÇÃO DE DADOS MUSICAIS

A mineração de dados musicais é uma área interdisciplinar que estuda métodos computacionais para entender e fornecer dados musicais e é um tópico de importância crescente com grande relevância comercial e potencial substancial. (LI, OGIHARA e TZANETAKIS, 2014)

Segundo os trabalhos de Li; Ogihara; Tzanetakis (2011) e Nierhaus (2009), as tarefas relacionadas à mineração de dados musicais podem ser das mais variadas e até mesmo compostas. Comumente passam por *visualization, information retrieval, summarization, clustering, classification, association* e outras tarefas mais avançadas.

Este campo de pesquisa, assim como muitos na ciência de dados, se beneficiou da evolução da tecnologia na era digital, vendo evoluir gradualmente o armazenamento digital, compressão de áudio, assim como aumentos significativos na largura de banda da rede, tornou a distribuição de música computacional uma realidade, de acordo com Li; Ogihara e Tzanetakis (2014). Desta forma, é evidente o avanço de modo a enfrentar o desafio de acessar e interagir efetivamente com coleções musicais cada vez maiores e dados relacionados, tais como estilos, artistas, letras e resenhas musicais de uma forma relevante e inovadora.

Os avanços discutidos se espalham para outros campos, visto que os dados musicais são de escala significativa com diversas fontes e aspectos diferentes de informação, segundo Nierhaus (2009). Muitos dos aspectos relevantes, tais como comportamentos dos usuários e recuperação de informações baseadas na web, têm analogias diretas em outros itens de conteúdo multimídia, incluindo imagens e vídeos, como visto no trabalho de Li; Ogihara e Tzanetakis (2014). Como resultado, as técnicas e ferramentas desenvolvidas na mineração de dados musicais podem ser relevantes e úteis em geral para outras áreas de pesquisa multimídia. (LI, OGIHARA e TZANETAKIS, 2014)

2.3.1 Pré-processamento

A fase de pré-processamento de dados é talvez a mais crucial no processo de mineração de dados (AGGARWAL, 2015). No contexto musical, mesmo com muitos conjuntos de dados "prontos" já disponíveis, ainda é uma prática extremamente comum e essencial. Isso ocorre, principalmente, pela infinidade de especificidades possíveis para se aprofundar em cada objetivo deste contexto. Normalização costuma ser a técnica mais utilizada, com etapas adaptadas caso a caso, segundo Li, Ogihara e Tzanetakis (2011). Estão descritos abaixo alguns casos de normalização para o contexto de música computacional.

Um exemplo relativamente vetusto é o de um sistema capaz de fornecer informação bibliográfica para uma sequência de notas musicais de entrada, utilizando uma base de dados musicais normalizados. O sistema em questão reproduz uma peça musical armazenada ao encontrar uma correspondência entre a peça armazenada e uma *string* de notas musicais inseridas (HOFFBERG, 1999). Neste caso, cada conjunto de sequência de notas armazenadas na base de dados é normalizado em relação a um mesmo grau de escala, ou seja, a primeira nota de cada sequência de referência é sempre da mesma altura, por exemplo, em uma frequência de 220 cps. A sequência de entrada é normalizada também em relação ao mesmo grau de escala, de modo a realizar a consulta. (HOFFBERG, 1999)

Para o problema de classificação abordado por Wülfing e Riedmiller (2012), *patches* extraídos de sinais de áudio passam por normalização e *whitening transformation,* fruto dos estudos de Oja e Hyvarinen (2000). Essa etapa se mostra crucial para garantir uma boa qualidade das respostas das características aprendidas no treinamento.

Na tarefa de geração, Yamshchikov e Tikhonov (2020) realizaram duas etapas principais de pré-processamento, separação dos arquivos MIDI em faixas, priorizando as de maior importância, e normalização. A normalização foi especialmente aplicada nas notas e pausas dos arquivos MIDI. Ainda na tarefa de geração, Wei, Wu e Su (2019) normalizaram os sinais dos áudios sintetizados de arquivos MIDI em 44,1 kHZ e o tempo em 120 BPM. Em outro estudo abordado por Hadjeres (2018), todas as peças monofônicas soprano do conjunto de dados Bach Chorale do Music21, apresentado por Cuthbert e Ariza (2010), foram extraídas e codificadas na representação melódico-rítmica descrita por Hadjeres, Pachet e Nielsen (2017). Sobre a aplicação desta codificação, Hadjeres comenta:

> Nesta codificação, o tempo é quantificado usando uma décima sexta nota como a mais pequena subdivisão (cada batida é dividida em quatro partes iguais). Em cada uma destas subdivisões, o nome real da nota é utilizado como um *token* se for a subdivisão em que a nota é tocada, caso contrário, é utilizado um *token* adicional, denominada "__", de modo a indicar que a nota atual é mantida. (HADJERES, 2018, p. 100)

2.3.2 Ferramentas de mineração de dados musicais

2.3.2.1 Music Visualization

Music Visualization, ou visualização musical, se divide em duas categorias: (I) as que se concentram na visualização do conteúdo de metadados ou conteúdo acústico de documentos musicais individuais; e (II) as que visam representar coleções completas de música para mostrar as correlações entre diferentes peças musicais, ou agrupar peças musicais em diferentes agrupamentos com base em suas semelhanças de pares (LI, OGIHARA e TZANETAKIS, 2014). O primeiro caso tem como alvo um usuário casual, querendo ter uma perspectiva maior sobre a música que está prestes a escutar; já o segundo visa ajudar usuários a encontrar canções particulares que lhes interessam, como um sistema de recomendação por representação visual, segundo Li, Ogihara e Tzanetakis (2011).

2.3.2.2 Music Information Retrieval

Este é um campo em ascensão cujo foco é servir os usuários em relação à informação musical, conseguindo enganar por ser de fácil acesso e divertido. Essa informação pode ser representada por facetas não mutuamente exclusivas. Essas facetas são: altura, temporal, harmônica, timbral, editorial, textual, e bibliográfica (DOWNIE, 2003). Grande parte dos estudos considera um de dois contextos, baseados em conteúdo ou descritos por um conjunto de características diretamente computadas a partir de seu conteúdo, segundo Li, Ogihara e Tzanetakis (2011).

Uma grande variedade de diferentes métodos de busca de conteúdo em partituras e dados de áudio foi proposta e implementada em protótipos de pesquisa e sistemas comerciais. Por exemplo, a identificação de instâncias de gravações deve ser baseada em dados de áudio, enquanto os trabalhos são melhor identificados com base em uma representação simbólica. (LI, OGIHARA e TZANETAKIS, 2014)

2.3.2.3 Association Mining

Association Mining, ou Mineração associativa, identifica correlações entre instâncias em um conjunto de dados, segundo Li, Ogihara e Tzanetakis (2011). Também conforme o livro de Li, Ogihara e Tzanetakis (2011), ela pode ser dividida em três categorias: (i) detecção de associações entre diferentes características acústicas; (ii) detecção de associações entre música e outros formatos de documentos; e (iii) detecção de associações entre características musicais e outros aspectos musicais, por exemplo, as emoções.

Para detectar associações entre características musicais, Xiao et al. (2008) buscaram entender a relação entre timbre e tempo com modelos estatísticos, visando utilizar informações timbrais para melhorar o desempenho da estimativa de tempo. Para detectar associações entre formatos diferentes, Knopke (2004) usa o texto público visível e arquivos ocultos de aúdio em uma página na Web para metrificar sua similaridade. Para detectar associações entre características musicais e outros aspectos como emoções, Kuo et al. (2005) extrai informações musicais e altera um gráfico de afinidade de acordo, podendo fornecer uma percepção para as recomendações musicais, segundo Li, Ogihara e Tzanetakis (2011).

2.3.2.4 Classification

Classification, ou classificação, é um campo importante e de diversos novos desafios em anos recentes. Este contexto pode ser aplicado a vários casos, desde classificar peças até obras em diferentes grupos, a mais comum sendo a classificação do gênero/estilo musical. Além destas, existem classificações como de artista/cantor, classificação de humor/emoção, classificação de instrumentos, e assim por diante, de acordo com Li, Ogihara e Tzanetakis (2011).

Na classificação de áudio, costuma ser utilizado o processamento de voz e vídeo visando identificar e rotular o áudio em três classes diferentes: fala, música e som ambiente; segundo Li, Ogihara e Tzanetakis (2011). Porém, a abundância de dados de áudio necessários para alcançar taxas de classificação consideráveis acaba se tornando um detalhe pertinente, como visto no estudo de Foote (1997).

Para classificação de gênero musical, o objetivo é rotular corretamente o estilo de determinada música. As gravações podem ser organizadas hierarquicamente na coleção de gêneros e subgêneros (LI, OGIHARA e TZANETAKIS, 2011). Tzanetakis e Cook (2002) utilizaram um conjunto de dados enviesado para música clássica e jazz no seu estudo, com classificação baseada em características rítmicas, de tom e timbres. A falta de concordância na seleção do gênero é um problema típico da classificação musical, pois a relevância das diferentes categorias é extremamente subjetiva, assim como as próprias categorias. (LI, OGIHARA e TZANETAKIS, 2011)

Determinados padrões musicais representam contentamento ou relaxamento, enquanto outros fazem com que um indivíduo se sinta ansioso ou frenético, segundo Li, Ogihara e Tzanetakis (2011). Esta categoria de fenômeno gera incentivo para identificar e classificar emoções em músicas. Uma estrutura hierárquica com objetivo de automatizar a detecção de humor em dados musicais acústicos, considerando teorias psico-musicais nas culturas ocidentais, foi desenvolvida por Liu, Hu e Zhang (2003).

Uma possível preocupação em relação à classificação de emoções em músicas é a diversidade de sentimentos que seres humanos diferentes podem ter ao ouvir uma determinada peça, especialmente caso ela tenha um arranjo complexo entre melodias e letra. Esse ponto foi abordado por Werbock (2019), que afirmou por experiências próprias com indivíduos de diversas origens culturais que certos intervalos de tons e escalas geram sempre as mesmas respostas emocionais. Ainda assim, levantou como a preferência de cada indivíduo pode afetar o ato de desfrutar do efeito de um determinado intervalo de tom ou sequência de intervalos de tom, bem como que as respostas universais não significam que todos vão gostar daquilo que sentem quando os ouvem. De todo modo, a conclusão é relacionada ao conceito chamado resolução harmoniosa e como ela pode ser adiada ou alcançada através de certas melodias.

A detecção e classificação de instrumentos musicais tem dois objetivos principais: rotular gravações monofônicas como "amostras de som"; ou indexar os principais instrumentos incluídos em uma mistura musical, como separar uma guitarra no meio de uma música, de acordo com Li, Ogihara e Tzanetakis (2011).

Classificação de artista e identificação do cantor são problemas parecidos, mas não exatamente idênticos. Por um lado, a classificação de artista foca na rotulação de músicos com base nas características acústicas ou na voz do cantor. Por outro, a identificação do cantor permite o gerenciamento eficaz de bases de dados musicais com base na "similaridade do cantor", segundo Li, Ogihara e Tzanetakis (2011).

2.3.2.5 Clustering

Clustering, ou clusterização, especificamente no contexto de dados musicais, separa um conjunto de dados em grupos de objetos semelhantes considerando características similares de pares sem etiquetas de classe pré-definidas, segundo Li, Ogihara e Tzanetakis (2011).

Tsai, Rodgers e Wang (2004) examinam a viabilidade de agrupamento não supervisionado de dados acústicos de canções com base nas características de voz do cantor, extraídos via detecção de segmento vocal seguido de modelagem de sinal vocal solo. (LI, OGIHARA e TZANETAKIS, 2011)

2.3.2.6 Summarization

Para o contexto musical, *summarization*, ou sumarização, define os principais temas de uma determinada peça musical, podendo ser aplicada como uma representação da música que pode ser facilmente reconhecida pelo ouvinte, segundo Li, Ogihara e Tzanetakis (2011). Desta forma, se torna a representação que melhor resume o conteúdo digital de um objeto, sendo uma peça chave na organização e compreensão da informação em larga escala, como visto no estudo de Shao et al. (2004).

2.4 TEORIA MUSICAL

Segundo Pilhofer e Day (2019), a teoria musical foi consequência da música. A música existiu muito antes da teoria, que surgiu para explicar o que as pessoas estavam tentando realizar quando tocavam seus instrumentos. A teoria musical, em geral, reflete as regras gramaticais da linguagem de comunicação escrita, que surgiu após as pessoas terem obtido sucesso em falar umas com as outras.

O principal motivador e objetivo para a teoria musical foi a transcrição da música, permitindo assim outros músicos a ler e tocar composições de acordo como o compositor pretendia. Assim sendo, teoria musical não se torna necessária para a composição, mas pode ajudar os músicos a dominar novas técnicas, executar estilos de música desconhecidos e desenvolver a confiança de que precisam para tentar coisas novas, como descrevem Pilhofer e Day (2019).

2.4.1 Semitom

Considerando um teclado de piano como referência, se uma tecla for tocada e depois a tecla anterior ou seguinte for tocada, seja branca ou preta, a diferença desses sons será de meio-tom, ou um semitom, de acordo Pilhofer e Day (2019). Esse conceito pode ser visualizado na Figura 2.

Na notação musical ocidental, a menor diferença entre duas alturas é o semitom. Ainda assim, muitos outros sons microtonais estão presentes entre os semitons consecutivos.

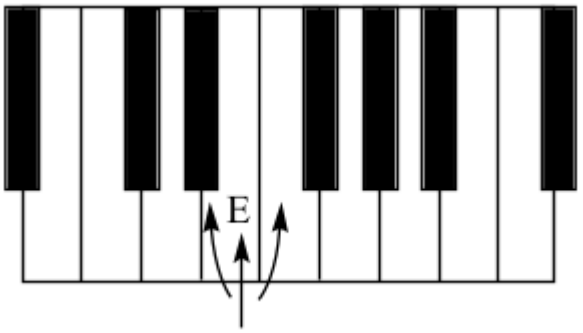


Figura 2 — Semitons identificados à esquerda e à direita da tecla E no piano.
(PILHOFER e DAY, 2019)

2.4.2 Sustenido

Na teoria musical, o sustenido representa o aumento da altura de determinada nota em um semitom. (MILLER, 2005)

O sustenido pode causar outras modificações em notas. Por exemplo, caso o sustenido marque uma nota dentro de um compasso, todas as ocorrências dessa nota dentro desse compasso devem ser aplicadas em sustenido, como visto na obra de Miller (2005).

2.4.3 Bemol

Inversamente ao sustenido, o bemol representa a diminuição da altura de determinada nota em um semitom, de acordo com Miller (2005).

Da mesma forma, o bemol causa modificações similares em notas. Como exemplo semelhante, caso marque uma nota dentro de um compasso, todas as ocorrências dessa nota dentro desse compasso devem ser aplicadas em bemol, segundo Miller (2005).

2.4.4 Altura

Para introduzir o conceito de altura, a primeira informação necessária é que as notas musicais são nomeadas com as primeiras sete letras do alfabeto — A, B, C, D, E, F, G — segundo Clendinning e Marvin (2016). A escala Dó-Ré-Mi-Fá-Sol-Lá-Si, é representada na seguinte ordem: C-D-E-F-G-A-B. Imagine estas sete letras subindo como escadas ou dispostas ao redor de um círculo como um relógio. "Contar" para cima ou para baixo na série recitando as letras para frente (sentido horário) ou para trás (sentido anti-horário). Para contar para cima além de G, começar de novo com A; para contar para baixo abaixo de A, começar de novo com G. (CLENDINNING e MARVIN, 2016)

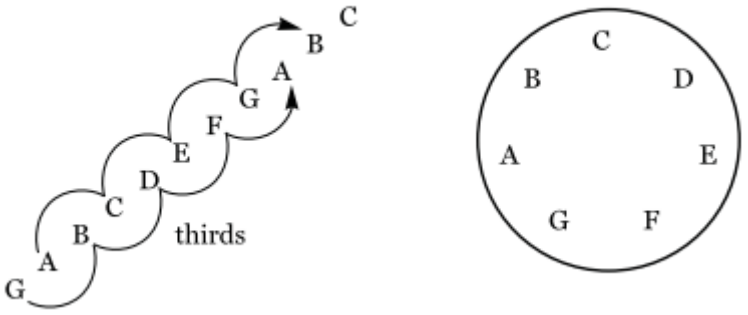


Figura 3 — Sete letras

(CLENDINNING e MARVIN, 2016)

Neste sistema, a letra se repete a cada oito posições. Por exemplo, na oitava posição após um C, estará outro C. Notas em um intervalo de oito letras formam uma oitava, estas que têm som semelhante graças à equivalência de oitavas. Notas em uma oitava são da mesma classe de altura e tem a mesma letra. Por exemplo, a classe de altura D representa cada D em uma oitava. Um tom, por outro lado, é aquela nota que soa em uma oitava em particular. (CLENDINNING e MARVIN, 2016)

2.4.5 Escala

Escalas são coleções de notas ordenadas, como visto no livro de Clendinning e Marvin (2016). Estas possuem um tom inicial e uma ordem para as notas que correspondem ao alfabeto musical. Uma escala maior, especificamente, pode começar em qualquer tom e precisa incluir cada uma das sete letras.

A palavra "escala" vem do latim *scalae* (ou *scala*, do italiano), que significa "escada". Neste caso, cada passo da escala é chamado grau ou degrau da escala. Quando você escreve ou toca uma escala, sua nota inicial, chamada tônica, é geralmente repetida uma oitava mais alta no final. (CLENDINNING e MARVIN, 2016)

2.4.6 Tríade

As tríades consistem em três alturas e são a categoria de acorde mais comum usada na música (PILHOFER e DAY, 2019). Seres humanos, providos de cinco dedos e alcance de quase uma oitava em cada mão, acabam tendo acordes com três notas consideravelmente espaçadas como unidade básica da harmonia ocidental, segundo Pilhofer e Day (2019).

2.4.7 Intervalo

Intervalo é a diferença de espaço e altura entre duas notas. Os intervalos são identificados por seu tamanho (normalmente um número entre 1 e 8) e qualidade (como maior ou menor). (CLENDINNING e MARVIN, 2016)

Para nomear intervalos, é importante contar a primeira e última letra, como descrito por Clendinning e Marvin (2016). Por exemplo, de A até D é um quarto (A-B-C-D) e de A até D é um quinto (A-G-F-E-D).

Intervalos melódicos são formados entre dois lances sucessivos em uma linha melódica. Os intervalos harmônicos são formados entre dois lances que soam ao mesmo tempo. (CLENDINNING e MARVIN, 2016)

2.4.8 Batida

Uma batida é uma pulsação que divide o tempo em comprimentos iguais (PILHOFER e DAY, 2019). Podemos pensar em um relógio como exemplo, em que a cada minuto, o ponteiro dos segundos faz tic-tac 60 vezes, e cada um desses tiquetaques é uma batida. É possível alterar o tempo da batida, aumentando ou diminuindo a velocidade do ponteiro, de acordo com Pilhofer e Day (2019).

2.4.9 Compasso

Compasso é a divisão de um trecho musical em séries regulares de tempos. É o agente métrico do ritmo (MED, 1996). Compasso também pode ser chamado medida ou barra. Costuma ser utilizado por compositores, visando separar suas obras em seções que ajudem os músicos a executar as peças como esperado, como visto no livro de Clendinning e Marvin (2016). Dessa forma, os músicos podem focar em uma parte por vez, aprimorando o desempenho final. Tipicamente, uma peça consiste em vários compassos de um mesmo comprimento.

2.4.10 Timbre

Timbre (cor do tom ou qualidade do tom) é o que diferencia dois sons da mesma frequência (mesma nota). Por exemplo, a nota em Dó tocada no violão soa muito diferente da nota em Dó tocada no piano ou na flauta. Isto significa que estes instrumentos têm timbres diferentes. (SIMPLIFYING THEORY, 2014)

Da mesma forma que o timbre é utilizado para diferenciar instrumentos diferentes, também pode ser aplicado para diferenciar cantores ou extensões vocais diferentes, como determinado por Pilhofer e Day (2019).

2.4.11 Intensidade

Podemos considerar a potência do som como intensidade ou sonoridade. É uma característica sonora que possibilita a diferenciação entre som de baixa intensidade e som de alta intensidade, baseado na descrição de Clendinning e Marvin (2016). A intensidade tem influência da energia de vibração da fonte que emite as ondas sonoras. As ondas sonoras, quando se propagam, costumam transmitir energias que se espargem em todas as regiões. Quanto mais potente é a energia transportada pela onda, maior é a intensidade do som que o ouvido capta. A intensidade é similar ao que costumamos chamar volume, visto no livro de Pilhofer e Day (2019).

2.4.12 Duração

Duração é a duração do tempo em que uma altura, ou tom, é tocada (BENWARD e SAKER, 2003). A duração é fortemente determinada pela categoria do fragmento musical em questão, visto que uma nota simples pode durar menos de um segundo e uma sinfonia pode durar mais de uma hora.

A duração costuma ser classificada como longa, curta, ou tomando um determinado período. Comumente ela é descrita com base nos termos de batida, como explicado por Delone et al. (1975).

2.4.13 Ritmo

O ritmo pode ser considerado como a recorrência de notas e pausas no tempo, formando um padrão rítmico caso se repita. Além de definir quando notas são tocadas, também estipula por quanto tempo e com que intensidade. Martineau define o ritmo da seguinte forma:

> O ritmo é o componente da música que pontua o tempo, levando de uma batida para a outra, e se subdivide em relações simples como o tom. Mesmo em ritmos aparentemente complexos, uma estrutura subjacente baseada em agrupamentos de divisões em 2 e 3 é muitas vezes perceptível. [...] As estruturas rítmicas são organizadas em medidas com o propósito de notação, que denotam o tempo dividido em grupos de batidas. [...] Na maioria dos ritmos também existe um pulso de batidas fortes e fracas, ou partes fortes e fracas das batidas [...] para transmitir um movimento harmônico e reforçar o sentido de tonalidade. O desdobramento e a variação das tensões e liberações resultantes temporalmente é responsável por grande parte do poder emotivo e expressivo do ritmo. (MARTINEAU, 2008, p. 12)

2.4.14 Harmonia

Podemos definir harmonia como a junção de sons ouvidos em simultâneo, normalmente de uma forma agradável. Notas tocadas isoladamente podem trazer monotonia musical, por isso acordes harmoniosos tornam as peças muito mais interessantes. Novamente, Martineau discorre sobre o conceito:

> A tríade maior, que ocorre naturalmente na série harmônica como um par de terças (um maior, depois um menor, somado a um quinto), é a base da música tercial (acordes construídos em terças) em todo o mundo, sendo a quinta perfeita e a terça maior, após a oitava, os intervalos mais estáveis e ressonantes, derivados dos tons superiores. [...] Tríades maiores na primeira inversão têm dois intervalos menores [...]. O mesmo se aplica aos acordes menores, que em sua primeira inversão soam marcadamente maiores, já que dois de seus três intervalos são maiores. [...] Inversões conspiram para fortalecer ou enfraquecer a importância da raiz. Na posição de raiz, os intervalos fundamentais estão todos no lugar como na série de tons, a nota inferior recebendo a identidade do acorde construído sobre ela [...]. (MARTINEAU, 2008, p. 16)

2.4.15 Melodia

Melodia é uma sequência de sons em intervalos irregulares, sendo criada pela sucessão de tons através do tempo, assim como descrito por Martineau (2008). Tons ascendentes e descendentes reforçam a disparidade entre notas altas e baixas. Grandes saltos nos tons refletem uma mudança grandiosa, enquanto saltos curtos demonstram naturalidade e delicadeza.

Melodias comumente consistem tanto de saltos longos quanto curtos, dado que ambos deixam brechas ocupadas pelo seu determinado oposto, graças a complementação que um traz ao outro. A natureza contínua da melodia significa que quando as notas se desviam para longe, o ouvinte, seguindo o caminho para descobrir onde ela leva, gosta que elas permaneçam conectadas e retornem. (MARTINEAU, 2008)

Concluindo, Martineau escreve:

> [...] A expressividade de uma melodia vem em parte pela tensão e liberação das notas intermediárias da escala, sua colocação rítmica sobre uma batida forte ou fraca, intensificando ou diminuindo seu efeito. Às vezes uma melodia pode atuar como duas melodias, saltando para cima e para baixo, mantendo assim alternadamente dois fios independentes, cada um em seu próprio nível de tom (ou registro). [...] Pausas silenciosas, ou descansos (caesura) são essenciais para as melodias. Elas dão tempo para a respiração, reflexão e interação com a música. Os ouvintes esperam pelo próximo evento, suspensos, antecipando. Um descanso bem colocado em um tema pode ser um momento musical poderoso. (MARTINEAU, 2008, p.18)

2.5 MÉTRICAS DE AVALIAÇÃO

Esta seção levanta métricas para validar modelos musicais generativos. Para extrairmos as métricas, usamos as seguintes *features*: *Pitch Count* (PC); *Pitch class histogram* (PCH); *Average Pitch Interval* (PI)*; Note length histogram* (NLH) e *Average inter-onset-interval* (IOI).

Como explicam Yang e Lerch (2020), existem dois principais objetivos para a avaliação com estas *features*. Obter métricas absolutas, visando gerar intuições sobre as propriedades e características de um conjunto de dados gerado ou coletado. Bem como métricas relativas, pretendendo comparar dois conjuntos de dados, como um gerado e coletado, por exemplo.

Sobre essa metodologia, Yang e Lerch ainda explicam:

> [...] O método proposto não visa avaliar peças musicais no contexto da criatividade ao nível humano, nem modela a percepção estética da música. Ele aplica o conceito de avaliação multicritério de modo a fornecer métricas que avaliem as propriedades técnicas básicas da música gerada e ajudem os pesquisadores a identificar questões e características específicas tanto do modelo quanto do conjunto de dados [...]. (YANG e LERCH, 2020, p. 2)

De modo a adquirir métricas absolutas — que podem produzir intuições interessantes sobre as propriedades do conjunto de dados de treinamento e as qualidades generativas da aplicação — a média e o desvio padrão de cada *feature* são calculados. Já as métricas relativas, que permitem comparar duas distribuições em várias dimensões, são calculadas aplicando validação cruzada para determinar a distância de cada amostra ao mesmo conjunto de dados (*intra-dataset* ou *intra-set*) ou ao outro conjunto de dados (*inter-dataset* ou *inter-set*). As saídas são em formato de histogramas de distância por *feature*, que tem então suas funções de distribuição de probabilidade (PDF) abalizadas por meio da estimativa de densidade por Kernel. Por fim, duas métricas são computadas, visando a avaliação objetiva de sistemas generativos considerando as PDFs das distâncias *intra-set* do conjunto de treinamento e *inter-set* entre os conjuntos de dados de treinamento e gerado: a área sobreposta (OA) e a divergência de Kullback-Leibler (KLD). Estas medidas de similaridade podem indicar o comportamento do sistema avaliado, pois ele compara a similaridade de dois conjuntos de dados de entrada um com o outro e dentro de si mesmos. (YANG e LERCH, 2020)

2.5.1 *Features* baseadas em Altura

2.5.1.1 *Pitch Count* (PC)

O número de diferentes alturas dentro de uma amostra MIDI. A saída é uma grandeza escalar para cada amostra. (YANG e LERCH, 2020)

2.5.1.2 *Pitch class histogram* (PCH)

O histograma da classe de altura é uma representação do conteúdo da altura independente da oitava, com uma dimensão de 12 para escala acromática [4,40]. Neste caso, ele representa a quantização cromática independente da oitava do continuum de frequência. (YANG e LERCH, 2020)

2.5.1.3 *Average pitch interval* (PI)

Valor médio de intervalos entre duas alturas consecutivas em semitons. A saída é uma grandeza escalar para cada amostra. (YANG e LERCH, 2020)

2.5.2 *Features* baseadas em Ritmo

2.5.2.1 *Note length histogram* (NLH)

O primeiro passo para extrair o histograma do comprimento de nota, ou NLH, é criar um conjunto de classes de comprimento de batidas permitidas, da seguinte forma: [completo, meio, quarto, 8º, 16º, ponto metade, ponto quarto, ponto 8º, ponto 16º, tercina de meia nota, tercina de semínima, tercina de colcheia], segundo Yang e Lerch (2020).

Sobre as características deste histograma, Yang e Lerch complementam:

> A opção de descanso, quando ativada, dobrará o tamanho do vetor para representar os mesmos comprimentos para as pausas. A classificação de cada evento é realizada dividindo a unidade básica no comprimento de (comprimento do compasso) / 96, e cada comprimento de nota é quantizado para a categoria de comprimento mais próxima. O vetor de saída tem um comprimento de 12 ou 24, respectivamente. (YANG e LERCH, 2020, p. 5)

2.5.2.2 *Average inter-onset-interval* (IOI)

Representa o intervalo de tempo entre o início (percebido) de uma nota e o da nota seguinte. Para calcular este intervalo no domínio da música simbólica, encontramos o tempo entre duas notas consecutivas. A saída é um escalar em segundos para cada amostra. (YANG e LERCH, 2020)

3 TRABALHOS RELACIONADOS

Visando encontrar artigos relacionados ao objetivo geral deste trabalho, foram colhidas pesquisas que, em algum nível do seu escopo, tinham preocupação em gerar acompanhamento musical para determinada entrada do usuário. Essa seção apresenta os dois artigos que melhor representam a ligação com este trabalho.

3.1 RL-DUET: ONLINE MUSIC ACCOMPANIMENT GENERATION USING DEEP REINFORCEMENT LEARNING

Jiang et al. (2020) tiveram como objetivo desenvolver um modelo capaz de criar música colaborativamente com um músico humano, escutando a música tocada e gerando acompanhamento. Para isso, o modelo foi treinado com o conjunto de dados Bach Chorale do Music21, um kit de ferramentas para Musicologia Assistida por Computador criado pelo MIT (CUTHBERT e ARIZA, 2010). Do conjunto de dados, foram utilizados corais com quatro partes monofônicas, no formato SATB (Soprano, Alto, Tenor e Bass) como conjuntos de dados de treinamento e validação, com 327 e 37 corais respectivamente, segundo Jiang et al. (2020).

Para representar os conceitos de altura, duração e articulação, eles foram adaptados ao contexto. Alturas simbólicas MIDI foram utilizadas para codificar notas. O tempo é quantificado em décima sexta nota, sendo a duração mais curta das notas no conjunto de dados Bach Chorale. Para notas maiores, foram utilizados símbolos "hold" (de retenção) para codificar sua duração.

As alturas podem ser representadas por símbolos "single-hold" ou "multi-hold", usando um símbolo para toda a altura ou um separado para cada ocorrência. Por exemplo, uma nota C4 será codificada com quatro passos de tempo: [C4, hold, hold, hold] ou [C4, C4_hold, C4_hold, C4_hold], nas representações “single-hold” e “multi-hold”, respectivamente. A representação “single-hold” é mais comumente utilizada, mas as experiências do trabalho demonstram uma maior diversidade na música gerada pela representação “multi-hold”, segundo Jiang et al. (2020). A palavra “hold” representa a continuidade da duração de determinada altura.

Para representar a batida e as informações rítmicas adicionais da música, Jiang et al. (2020) utilizam uma lista de subdivisões para codificar a batida em formação, similarmente como acontece no estudo de Hadjeres, Pachet e Nielsen (2017). O tempo é quantificado com dezesseis notas, cada batida sendo subdividida em quatro partes.

O estudo ainda tem modelos de recompensa, considerados pelos autores como a chave para o sucesso do algoritmo. Estes modelos consideram tanto a consistência temporal horizontal (BRIOT, HADJERES e PACHET, 2017) sobre a parte gerada pela máquina, quanto as relações de harmonia vertical (BRIOT, HADJERES e PACHET, 2017) entre as partes humana e máquina (JIANG et al., 2020). A consistência temporal horizontal está relacionada diretamente à duração das peças, ou seja, a consistência ao longo da sequência temporal. Já a harmonia vertical é relacionada à disposição e combinação de notas em uma determinada etapa temporal. Dessa forma, podemos considerar o eixo horizontal como o eixo temporal e o vertical como harmônico, como apresentado por Briot, Hadjeres e Pachet (2017).

Para validar o projeto, Jiang et al. (2020) comparam o desempenho do RL-Duet com um modelo de base *maximum likelihood estimation* (MLE) e um modelo de aprendizado por reforço com base em regras. Este modelo com base regras nada mais é que uma implementação própria do SequenceTutor, de Jaques et al. (2017), que usa uma recompensa baseada em modelos, com muitas recompensas baseadas em regras, e os pesos entre elas são ajustados com precisão (JIANG et al., 2020).

Foi observado por Jiang et al. (2020) que existe uma ligeira degeneração da parte gerada por todos os três algoritmos em comparação, considerando que o início da parte gerada sempre foi inicializado com o valor de referência. Ou seja, todos os algoritmos apresentaram algum nível de desvio nos resultados, considerando os valores de entrada. De todo modo, o RL-Duet se mostrou o mais robusto e o que alcança a métrica de avaliação mais próxima do valor de referência. Interessantemente, também foi descoberto que o MLE captura a tendência dessas métricas do conjunto de dados. Entretanto, devido ao viés de exposição (bias), os problemas de geração tendem a se acumular temporalmente, levando a desvios significativos e divergências das métricas objetivas.

Para avaliação subjetiva, foi aplicado um pequeno questionário e comparações em pares com usuários voluntários. É comparado o desempenho do RL-Duet com o modelo de base MLE, visto que o modelo RL-Rules está excluído devido à geração de resultados nada satisfatórios. O questionário possui as seguintes perguntas:

1. Você aprendeu a tocar um instrumento musical (não menos que 5 horas por semana) por mais de 5 anos no total no passado?
   - Sim
   - Não
2. Quanto tempo você gasta para ouvir música clássica a cada semana?
   - Menos de 1 hora
   - Entre 1 e 3 horas
   - Entre 3 e 5 horas
   - Mais de 5 horas

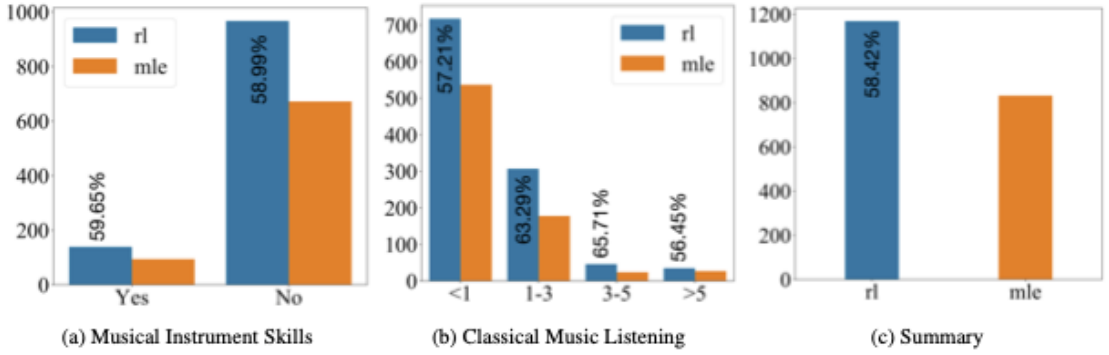


Figura 4 — Resultados da avaliação subjetiva

(JIANG et al., 2020)

Nas comparações, cada usuário recebeu um par de trechos curtos de dueto com um humano. Então, os trechos são randomicamente truncados se baseando em duetos gerados. Após ouvir cada par de amostras de dueto, o participante foi solicitado a escolher qual era o preferido. Cada par de amostras de dueto foi avaliado por 20 sujeitos. 125 sujeitos realizaram o teste e 2000 votos válidos foram coletados. 19 deles (com 233 votos) aprenderam um instrumento musical antes, e 28 deles (com 617 votos) passaram mais de 1 hora ouvindo música clássica por semana (JIANG et al., 2020).

Observando a Figura 4, podemos identificar que quando o nível de familiaridade com a música clássica é maior, a distinção entre o modelo RL-Duet e MLE é mais óbvia. No total, cerca de 58,42% dos sujeitos preferem os duetos gerados pelo RL-Duet aos gerados pelo modelo MLE (JIANG et al., 2020). Se os duetos em comparação se baseiam na mesma parte

humana, pode ser difícil diferenciar os dois duetos se o sujeito tem um histórico musical limitado. Já a preferência pelo RL-Duet em relação ao MLE é mais nítida para sujeitos com mais desempenho e experiências auditivas.

3.2 POPMAG: POP MUSIC ACCOMPANIMENT GENERATION

Ren et al. (2020) propuseram um modelo Transformer modificado com uma nova representação MIDI, que permite gerar múltiplas faixas simultaneamente em uma única sequência e modelar explicitamente a dependência das notas de diferentes faixas, visando melhorar a harmonia entre a entrada e o acompanhamento gerado para músicas Pop. Apesar de que isto melhore muito a harmonia, acaba consequentemente aumentando a duração da sequência e levantando o desafio da modelação musical de longo prazo. Quanto maior se tornam as peças, mais complexo e oneroso se torna o processo de manter a harmonia constantemente.

Acompanhamentos musicais costumam ser compostos por múltiplos instrumentos/faixas em forma de arranjo, para melhor expressividade. A geração considerando esse contexto pode ser chamada geração multi-faixa. Garantir a harmonia entre notas musicais em diferentes faixas é um problema chave. A representação MIDI criada para possibilitar uma solução do problema se chama MUlti-track MIDI (MuMIDI), que codifica eventos MIDI multi-faixa em uma sequência de tokens. Como o MuMIDI permite a geração de multi-faixas em uma única sequência, a dependência entre as notas musicais em diferentes faixas pode ser melhor capturada e mais informação pode ser aproveitada para melhorar a harmonia.

Para a música ser coerente e interessante, a estrutura de longo prazo — considerando o tamanho e duração das peças — se torna extremamente importante. Isto se torna especialmente relevante visto que o MuMIDI gera múltiplas faixas em uma única sequência, o que como efeito torna consideravelmente mais difícil modelar músicas a longo prazo. Visando uma solução, a duração da sequência foi encurtada em casos específicos, bem como o modelo Transformer-XL de Dai et al. (2019) foi adotado para o codificador e decodificador.

O PopMAG tem assim duas partes essenciais: Representações MuMIDI e um modelo aperfeiçoado sequência-a-sequência. Foram utilizados três diferentes conjuntos de dados de música pop e aplicados tanto testes objetivos quanto subjetivos para avaliação. Os resultados provenientes dos três conjuntos de dados foram promissórios e a aplicação tem um desempenho amplamente superior ao dos sistemas de geração de acompanhamento musical do estado da arte.

Como comentado anteriormente, garantir a harmonia entre múltiplas faixas é importante para gerar acompanhamentos musicais de alta qualidade. Desta forma, Ren et al. (2020) engendraram a nova representação MuMIDI para codificar notas musicais multi-faixa em uma única sequência compacta de tokens. Para codificar uma peça musical multi-faixa com estrutura de dados complexa em uma única sequência de símbolos, são utilizados alguns símbolos incluindo compassos, posição, faixa, nota, acorde e meta símbolos. O início de uma nova posição na peça musical é representado pelos símbolos de compasso e posição juntos, seguido pelo início de uma nova faixa com o símbolo de faixa ou um símbolo de acorde denotando o acorde que as notas subsequentes devem seguir, e, por fim, símbolos de nota são adicionados.

Cada compasso é dividido em 32 timesteps, definindo o tempo de início de cada nota para o timestep mais próximo. O símbolo de posição é seguido de um símbolo de faixa, representando a faixa à qual pertencem as notas subsequentes. A faixa pode simbolizar uma melodia, bateria, piano, corda, guitarra ou contrabaixo.

Em relação às notas, diferentemente das demais representações, os autores optaram por definir todos os atributos de uma nota (altura, velocidade e duração) em um único símbolo, visando uma simbologia menos verbosa. Então, ao prever múltiplos atributos de uma nota musical, múltiplas matrizes softmax foram adicionadas na saída oculta para gerar atributos correspondentes a esta nota. Ao longo do trabalho de Ren et al. (2020), para acordes, foram consideradas 12 notas raízes (C, C#, D, D#, E, F, F#, G, G#, A, A# e B) e 7 categorias de acorde (maior, menor, diminuto, aumentado, maior7, menor7 e meio_diminuto).

Como visto no trabalho de Ren et al. (2020), o tempo é o único metadado tratado pelo MuMIDI, podendo ser categorizado em baixo (inferior a 90), médio (90 a 160) e alto (superior a 160) e sendo representado por três meta-símbolos.

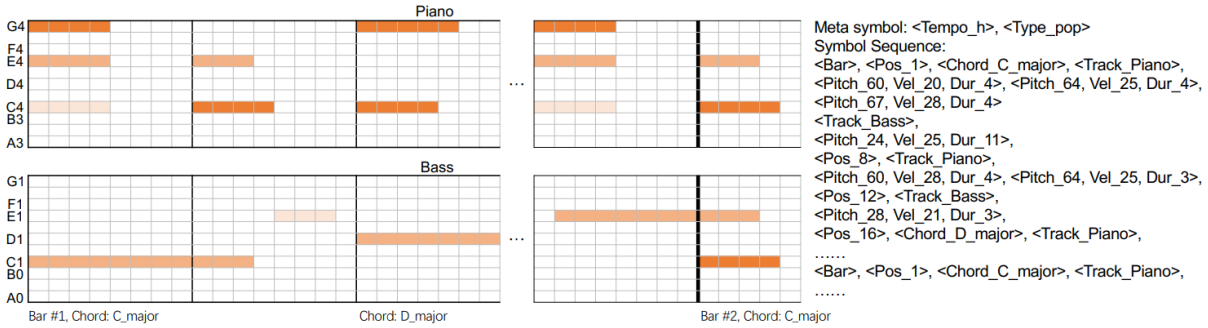


Figura 5 — Exemplo de sequência de tokens MuMIDI convertidos a partir de um segmento de partitura musical

(REN et al., 2020)

A arquitetura Transformer-XL de Dai et al. (2019) é baseada em Transformer e pode aprender dependências muito mais longas que redes neurais recorrentes (RNNs), por exemplo, além de gerar peças de texto suficientemente coerentes com milhares de tokens. Por conta disso, foi selecionado para servir de base ao codificador e decodificador.

O módulo de entrada do modelo é usado para transformar símbolos MuMIDI em representações de entrada, que consistem de incorporações de compasso, posição e token. As incorporações de compasso e posição lidam com notas no mesmo intervalo de tempo igualmente, bem como distinguem as notas em diferentes intervalos de tempo.

O módulo de saída esboça diretamente as saídas do decodificador três vezes para obter três logits diferentes. Após, a função softmax é aplicada para obter três distribuições categóricas de probabilidade baseadas nos atributos das notas. Se o símbolo da etapa em questão não for uma nota, apenas a primeira distribuição é utilizada como saída.

Para o treinamento, foi utilizada a técnica Teacher Forcing, fornecendo tokens com valor de referência para o decodificador gerar os tokens seguintes. A entropia cruzada entre eles é minimizada visando a otimização do modelo. Para inferência, os tokens alvo são gerados um a um, armazenando a incorporação de compasso/posição atual e a atualizando conforme o conteúdo gerado.

Foram utilizados três conjuntos de dados para avaliar o PopMAG, sendo eles: Lahk MIDI (LMD) apresentado por Raffel (2016), um subconjunto de música pop do FreeMidi e um conjunto de dados de pop chinês MIDI (CPMD). Grande parte destes dados foram gerados pelo usuário, tornando os dados ruidosos demais para serem usados diretamente. Dado este contexto, foram aplicadas etapas de limpeza como extração de melodia, compressão, filtragem e segmentação de dados e reconhecimento de acordes. Assim, cada conjunto de dados foi dividido em 3 partes: 100 amostras para validação, 100 amostras para testes e as demais para treinamento.

O modelo possui um codificador e um decodificador baseados em Transformer recorrente. O codificador é constituído por 4 camadas e 8 nodos, já o decodificador por 8 camadas e 8 nodos.

A geração de faixas de baixo, piano, guitarra, corda e bateria condicionadas à melodia e acorde é a tarefa padrão, com o número máximo de 32 compassos. Amostragem estocástica é utilizada para garantir diversidade no conteúdo gerado, bem como nos trabalhos de Huang et al. (2018) e Huang e Yang (2020).

Para avaliação subjetiva, foram 15 participantes avaliadores, sendo 5 destes capazes de compreender teoria musical básica. Foram gerados 100 conjuntos de peças, compostas por geradas e valores de referência, para serem avaliados. Cada um é avaliado por todos participantes e o com a melhor harmonia geral deve ser apontado. A média do total de votos de cada conjunto apresenta a pontuação de preferência final.

A avaliação objetiva é composta por métricas inspiradas por Huang e Yang (2020), Yang e Lerch (2020) e Zhu et al. (2018), sendo elas: Precisão do Acorde (CA), que mede se os acordes gerados se encaixam nos condicionais; Perplexidade (PPL), métrica comum em problemas de geração de texto para medir quão bem um modelo se adapta à sequência. A distribuição de algumas características também são usadas para avaliação, comparando com às dos valores de referência. Essas distribuições são calculadas por histogramas suavizados, sendo elas: Altura (P); Velocidade (V), quantificada em 32 níveis de velocidade; Duração (D), também quantificada em 32 atributos de duração e Inter-Onset Interval (IOI), que quantifica os intervalos em 32 classes da mesma forma que a duração da nota. Por fim, a área sobreposta (OA) média das distribuições (DA, onde A pode ser uma de P, V, D e IOI) é calculada para medir a diferença entre a peça musical gerada e a peça musical usada como valor de referência, acordo com Ren et al. (2020).

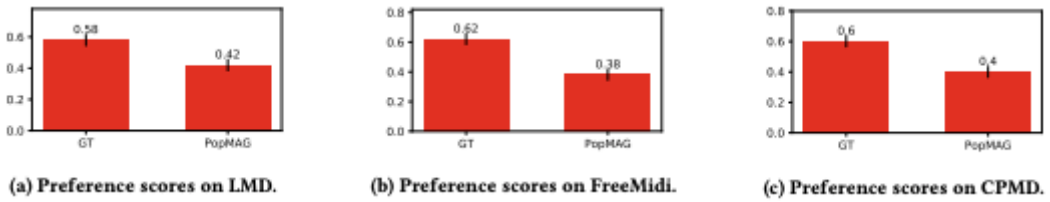


Figura 6 — Avaliações subjetivas do PopMAG no teste de melodia para outros tipos de tarefa. As barras de erro mostram os desvios padrão da média.

(REN et al., 2020)

Observando os resultados promissores do teste aplicado pelos autores para avaliar a harmonia geral e a qualidade das peças musicais gerados, é perceptível que, mesmo existindo uma diferença considerável entre peças compostas por humanos e geradas, cerca de 42%, 38%, 40% em três conjuntos de dados atingiram a qualidade dos valores de referência.

Para comparar o PopMAG com estudos semelhantes, foram simulados experimentos dos mesmos problemas com as mesmas características destes comparáveis. Um exemplo foi o MuseGAN de Dong et al. (2018), onde a comparação demonstrou que o PopMAG o supera em todas as métricas subjetivas e objetivas (REN et al., 2020), o que garante qualidade nas peças geradas e que a representação MuMIDI mantém a harmonia de maneira superior à representação baseada em pianoroll utilizada pelo MuseGAN.

Realizando mudanças necessárias para credibilidade e comparando diretamente MuMIDI com outras representações similares, sendo estas MIDI-like (HUANG et al., 2018) e REMI (HUANG e YANG, 2020), MuMIDI chegou a melhores pontuações do que MIDI-like e REMI, tanto subjetivas quanto objetivas.

Visando uma avaliação mais granular, ainda foi analisada a eficácia do método de modelagem em nível de nota. Com os resultados, é possível concluir que o MuMIMI consegue convergir e gerar uma peça musical mais rapidamente do que outros métodos de modelagem de representação MIDI, graças à menor sequência de tokens com modelagem em nível de nota (REN et al., 2020).

Ainda sobre análise, foi avaliada a memória no codificador e decodificador. Foram feitos testes com memória em ambos, com memória em apenas um e sem memória em ambos. Os resultados apresentaram todas métricas superiores para o PopMAG com memória tanto no codificador quanto no decodificador.

A última avaliação valida a eficácia das incorporações de compasso e posição. Para isso, são comparadas com incorporações sinusoidais (VASWANI et al., 2017) e relativas (DAI et al., 2019) de posição. Os resultados confirmam que a modelagem com MuMIDI pode capturar melhor a estrutura musical do modelo, segundo Ren et al. (2020).

Finalizando, são expostas ideias de trabalhos futuros com o PopMAG, como geração de acompanhamento musical granular e gerações controladas pela emoção e estilo (REN et al., 2020). Treinamento generativo em larga escala, como nos trabalhos de Brown et al. (2020) e Song et al. (2020), também foi considerado. Da mesma maneira, Ren et al. (2020) levantaram a possível aplicação do modelo em outras tarefas, como geração de progressão de acordes, geração de acompanhamento de voz de canto e classificação MIDI.

3.3 CONSIDERAÇÕES

A Tabela 1 faz uma comparação entre os dois trabalhos apresentados em relação às técnicas, conjuntos de dados e categorias de avaliação utilizadas.

Trabalho	Técnica de ML	Conj. de dados	Avaliação
RL-Duet	Apr. por reforço	Bach Chorale	Obj. e Subj.
PopMAG**	Apr. supervisionado	LMD*, CPMD e FreeMidi	Obj. e Subj.

Tabela 1 — Tabela comparativa das técnicas, conjuntos de dados e categorias de avaliação entre os trabalhos relacionados

No que lhe concerne, a Tabela 2 mede os dois trabalhos em concernência às métricas utilizadas por eles.

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
RL-Duet	+0.12	-0.48	-0.09	0.0057	0.042	-	-	-
PopMAG**	-	-	-	-	-	0.58±0.01	0.55±0.01	0.72±0.01

Tabela 2 — Tabela comparativa das métricas entre os trabalhos relacionados

** Métricas baseadas neste conjunto de dados*

*** Métricas da tarefa de piano para outras categorias de acompanhamento*

Já a Tabela 3 apresenta métricas diretamente relacionadas aos conjuntos de dados dos trabalhos relacionados.

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Bach Chorale	3.25	4.57	3.84	-	-

Tabela 3 — Tabela apresentando métricas dos conjuntos de dados dos trabalhos relacionados

4 TREINAMENTO DO MODELO E DESENVOLVIMENTO DO PROTÓTIPO

Um dos principais objetivos deste trabalho é validar a possibilidade de superar, ou ao menos equiparar, as métricas apresentadas nos trabalhos relacionados. Ademais, conforme descrito anteriormente, este estudo visa objetivamente gerar melodias de acompanhamento baseadas em batidas, diferentemente dos demais trabalhos relacionados que aplicaram métodos similares para outros fins. Para isso ser cumprido, a produção musical foi organizada em 7 etapas. São elas: definição do conjunto de dados, escolha, ajustes e treinamento do modelo, avaliação de resultados, ajustes de parâmetros e desenvolvimento do protótipo. Observando a Figura 7, é possível visualizar o fluxo dessas etapas.

Na primeira fase, foi realizada uma pesquisa e análise de diversos conjuntos de dados disponíveis online, muitos já utilizados em outros estudos, para selecionar o mais adequado ao escopo deste trabalho. Na etapa de escolha do modelo, foram testadas e analisadas as possíveis opções disponibilizadas pela biblioteca Magenta e selecionada a que mais se encaixava ao caso de uso deste estudo. Seguindo o fluxo, na parte de ajustes no modelo, foram aplicadas mudanças à implementação do modelo e ao processo de treinamento para se adequar ao contexto apresentado. Já na fase de treinamento, o modelo foi treinado com o conjunto de dados previamente determinado e consequentemente avaliado na etapa seguinte, de avaliação. Em sequência, na parte de ajuste de parâmetros, alguns foram adaptados visando melhorar o desempenho. Por fim, foi desenvolvido o protótipo com interface de usuário conforme proposto no objetivo do trabalho.

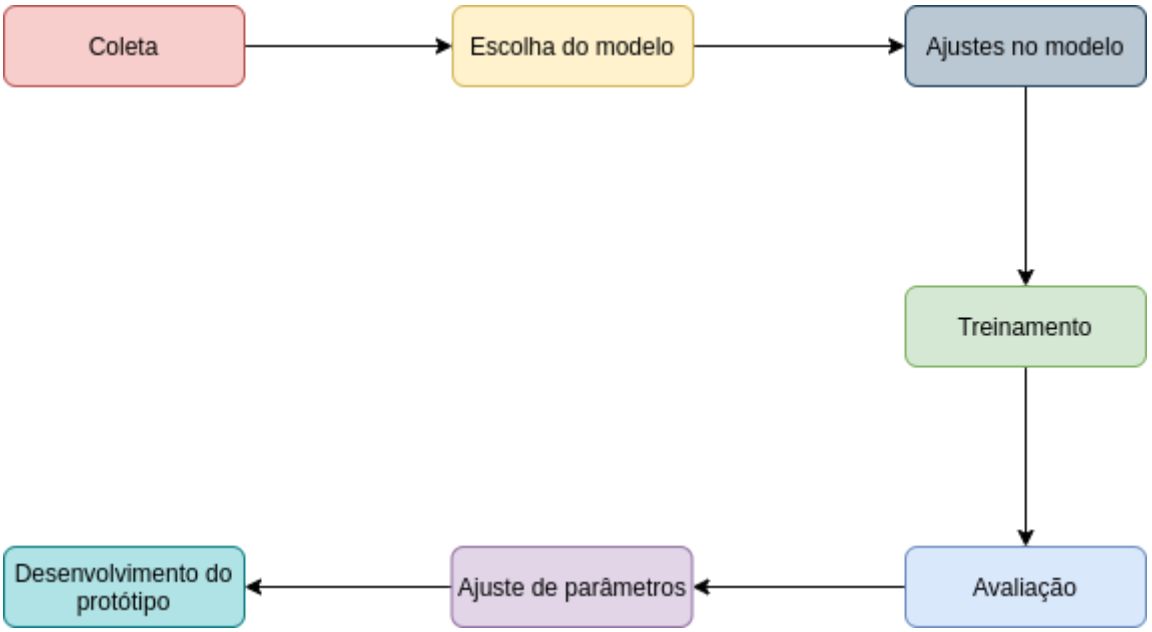


Figura 7 — Etapas de concretização do protótipo para geração das melodias de acompanhamento

4.1 FERRAMENTAS

O equipamento utilizado consistiu de um Notebook Acer F5-573G-50KS, que no que lhe concerne conta com um processador Intel Core i5 7200U, 8 GB de memória RAM DDR4 2133 MHz, placa de vídeo dedicada NVIDIA Geforce 940MX 2 GB GDDR5, HD de 1 TB, SSD de 120 GB, tela 15.6″ HD 1366 x 768 pixels Widescreen e sistema operacional Ubuntu 20.04. Além desse computador, foi utilizado um HD externo Toshiba de 1 TB para armazenar dados do projeto.

Para pleno funcionamento da placa de vídeo dedicada com as ferramentas, foi necessário instalar os drivers proprietários, bem como os pacotes CUDA e cuDNN, da empresa NVIDIA. CUDA é uma plataforma de computação paralela e um modelo de programação. Complementarmente, cuDNN permite que a comunidade de estruturas de redes neurais se beneficie igualmente de suas APIs. Assim, os usuários da cuDNN não são obrigados a adotar nenhuma estrutura de software específica, ou mesmo layout de dados. (CHETLUR, 2014)

Python foi a linguagem de desenvolvimento selecionada, esta que é uma linguagem de programação interpretada, de alto nível e pode ser usada para diferentes intuitos (KUHLMAN , 2011). Python tem código aberto e conta com 52 dissímeis bibliotecas relacionadas à ciência de dados, auxiliando ainda mais o desenvolvimento, segundo Grus (2015). Diversas das demais ferramentas e bibliotecas aplicadas neste projeto utilizam Python como linguagem base, como, por exemplo, TensorFlow e Magenta que serão descritas na sequência.

A biblioteca Python mais utilizada no desenvolvimento foi a Magenta. Magenta é um projeto de pesquisa da Google, que explora o papel da aprendizagem de máquinas como uma ferramenta no processo criativo. Ela inclui utilitários para manipulação de dados (principalmente música e imagens), usando estes dados para treinar modelos de ML, e finalmente gerando novos conteúdos a partir destes modelos. Assim como inúmeras ferramentas, utiliza o TensorFlow por trás dos panos. TensorFlow é outra biblioteca de código aberto para computação numérica e aprendizagem de máquinas em larga escala, aplicável a uma ampla variedade de tarefas. A versão utilizada do TensorFlow foi 2.4.1.

4.2 DEFINIÇÃO DO CONJUNTO DE DADOS

Nesta primeira etapa, a coleção de dados que mais se encaixava ao propósito deste estudo foi procurada, analisando e validando diversas coletâneas pré-existentes para aperfeiçoar a seleção com um maior embasamento. No início do processo, foram verificados os próprios conjuntos de dados disponibilizados pelos pesquisadores do projeto Magenta. Tudo considerado código aberto e de livre utilização.

Entre os conjuntos musicais dispostos, haviam dois relacionados à percussão, com atuações variadas de humanos tocando bateria; um relacionado ao piano, com uma gama de atuações virtuosas e um relacionado às notas musicais, com uma alta densidade de notas tocadas em diferentes instrumentos, cada uma com um tom e timbre únicos.

Com essa análise, foi possível compreender qual categoria de dados seria adequada para uso em conjunto com a biblioteca, em questão de estrutura das informações e de

variedade da matéria-prima. Esse conhecimento permitiu a expansão da busca para outros dados providenciados em diferentes estudos ou *websites*.

Baseado no mencionado acima, foi possível identificar que os modelos de acompanhamento já treinados pela equipe do projeto utilizaram dados relacionados ao instrumento de batida. Por exemplo, o modelo treinado para uso do plugin *Drumify*, responsável por gerar faixas de bateria para acompanhamento dada uma determinada entrada do usuário, foi treinado com um dos conjuntos de dados com atuações de bateria, como apresentado por Roberts et al. (2019)

Dessa forma, com o objetivo deste trabalho em mente, foi utilizada uma coletânea de dados com uma pluralidade de batidas. A procura levantou muitas opções relevantes, como alguns reunidos pela organização sem fins lucrativos *International Society for Music Information Retrieval*.

Por fim, a coletânea selecionada foi a Groove MIDI Dataset (GMD), constituída pela equipe responsável pela biblioteca *Magenta*. Como documentado por Gillick et al. (2019), Groove MIDI Dataset possui 13,6 horas distribuídas em 1.150 arquivos MIDI de batidas revisadas, aprofundadas e utilizadas em várias publicações. Uma versão expandida, com 444 horas de áudio, mais metadados e diversidade foi sugerida por Callender, Hawthorne e Engel (2020) com o nome de Expanded Groove MIDI Dataset (E-GMD). Para este trabalho, a versão original, GMD, de Gillick et al. (2019) foi utilizada, em uma opção disponibilizada mais leve, que traz exclusivamente os arquivos MIDI, sem outros tipos de arquivos como WAV.

Sessão	Estilo	BPM	Estilo de batida	Duração	Divisão
drummer1/eval_session	funk/groove1	138	beat	27.872.308	test
drummer2/session3	rock/shuffle	85	beat	203.414.604	train
drummer4/session1	latin/brazilian	184	beat	38.564.543	validation
drummer7/session1	hiphop	100	fill	10.276.250	train

Tabela 4 — Tabela com metadados selecionados de uma amostra do conjunto de dados GMD

Na Tabela 4, podemos analisar algumas informações de uma amostra de dados do conjunto GMD. A sessão demonstra qual o baterista responsável por aquela faixa, sendo enumerado, bem como qual tipo de sessão é, entre avaliação ou treinamento/validação; O estilo informa a qual gênero musical aquela determinada faixa pertence; BPM são o número de batidas por minuto; Estilo de batida, podendo ser uma batida (*beat*) ou outro tipo de som (*fill*); Duração seria a extensão da faixa em segundos; e Divisão significa a qual parte predefinida do conjunto a faixa se encaixa, podendo ser *train*, *validation*, ou *test*.

4.3 PRÉ-PROCESSAMENTO DE DADOS

Um arquivo de registro tensorflow é objeto tensorflow mantido em um arquivo e tratado como um iterador em sequências de notas (RUÉ VILÀ, 2020). O conjunto de dados deve ser convertido com o script *convert_dir_to_note_sequences*, conforme a Figura 8.

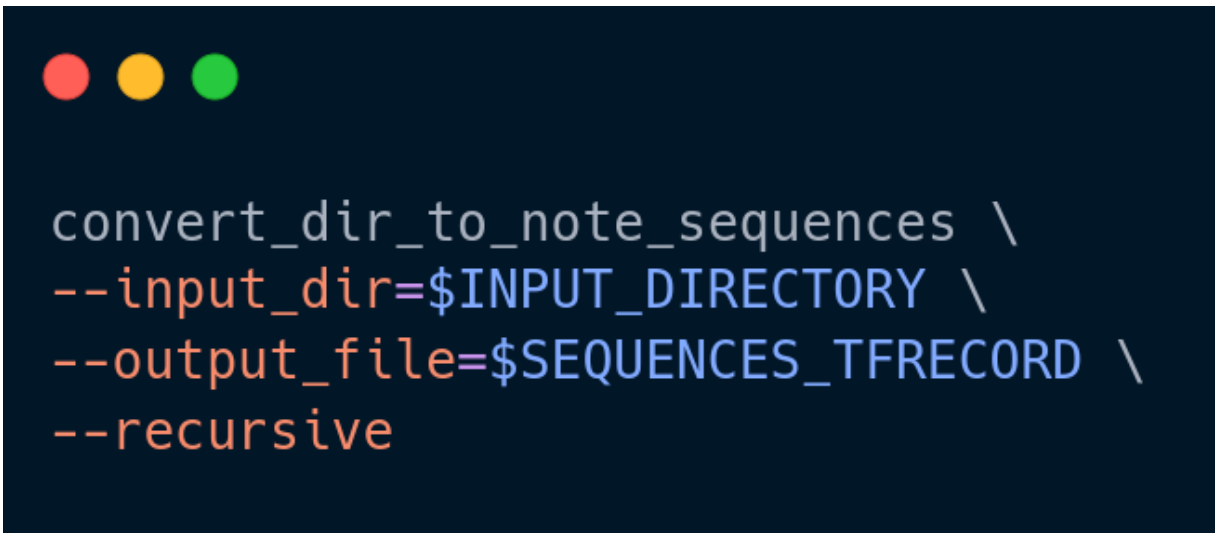


Figura 8 — Comando executado para converter o conjunto de dados em NoteSequences

A execução cria um arquivo de registro TF no caminho especificado. Dessa forma, o conjunto pode ser carregado pelo arquivo e aplicado no processo de treinamento. Com o TFRecord, é interessante particionar o mesmo para treinamento e processos de avaliação.

Para isso, foi utilizada a classe a RandomPartition da Magenta. O script é extraído do livro de Alexandre Dubreuil (2020). O comando foi executado como mostra a Figura 9:

```
python magenta/scripts/partitioner.py \
--config="groovae_2bar_melody_tap_fixed_velocity" \
--input="$DATASET_TFRECORDS_FILE" \
--output_dir="$DESIRED_DIRECTORY" \
--eval_ratio=0.3
```

Figura 9— Comando executado para dividir o conjunto de dados

Com isso, dois *tfrecords* são criados como eval.tfrecord e train.tfrecord no diretório de saída definido, com uma taxa de avaliação de 0,3. Ou seja, o conjunto de avaliação corresponde a 30% do conjunto de dados.

O train.tfrecord será usado no processo de treinamento, e o eval.tfrecord para avaliar as saídas dadas pelo modelo, visto que elas não foram consideradas no processo de treinamento.

4.4 ESCOLHA DO MODELO

Para escolha do modelo, foram testados e analisados diversos disponibilizados através da biblioteca Magenta, considerando o contexto deste trabalho e de projetos semelhantes, como outros plugins da Magenta Studio — vistos em Roberts et al. (2019) — ou os apresentados na seção de trabalhos relacionados.

O primeiro modelo cogitado foi o *Melody RNN*, modelo que utiliza uma rede neural recorrente que aplica modelagem de linguagem à geração de melodias usando memória de curto prazo longo, ou *long short-term memory* (LSTM). Esse modelo parece imediatamente relevante, dado seu claro objetivo de gerar melodias, que coincide com uma das realizações esperadas por este estudo. Além disso, possui diversas funcionalidades e variações

desenvolvidas visando aumentar a diversidade das obras geradas. Por exemplo, o modelo chamado *Attention RNN*, permite que acesse mais facilmente informações passadas sem ter que armazenar essas informações no estado da célula da RNN, conforme descrito por Waite (2016). Isto permite que o modelo aprenda mais facilmente dependências a longo prazo, e resulta em melodias com conteúdos mais longos.

Apesar de promissor por seu contexto de geração melódica, o modelo acima não pareceu ideal para gerar melodias como acompanhamento de outros instrumentos, visto que outro modelo mais adequado ao projeto foi encontrado, *MusicVAE*, descrito abaixo. Ademais, o Melody RNN não foi desenvolvido para receber entradas com sons de outros instrumentos diferentes daquele que fora treinado

O modelo *MusicVAE*, que utiliza um autocodificador hierárquico recorrente e variável para trabalhar com música, chamou a atenção. Este modelo é particularmente interessante pela sua versatilidade, pois pode ser utilizado em diversos contextos. O *MusicVAE* aprende um espaço latente de sequências musicais, fornecendo diferentes modos de criação musical interativa, como amostragem aleatória a partir da distribuição prévia; interpolação entre sequências existentes ou manipulação dessas através de vetores de atributos, ou de um modelo de restrição latente, segundo Roberts et al. (2018). Mais alguns detalhes sobre o modelo podem ser observados abaixo:

> [...] Para sequências curtas, utiliza um codificador bidirecional LSTM e um decodificador LSTM. Para sequências mais longas, é aplicado um novo decodificador LSTM hierárquico, que ajuda o modelo a aprender estruturas de longo prazo. (ROBERTS et al., 2018)

Com essas informações, foi possível perceber que esse modelo poderia se alinhar com o desejado. Estudando o *MusicVAE* mais profundamente, foi encontrada uma variação desse modelo, chamada *GrooVAE*. Ela visa gerar e controlar atuações expressivas de bateria, facilitando variados desafios de produção. Foi utilizada com o conjunto de dados de bateria da biblioteca Magenta para treinar o modelo que fomenta o plugin *Druminfy*, mencionado no capítulo 4.2.

Para validar a possibilidade do uso do *MusicVAE*, foram aplicadas sessões de treinamento com o conjunto de dados definido no capítulo 4.2. Houveram alguns resultados errôneos, mas aparentou que, com alguns ajustes, o modelo poderia trazer os valores esperados.

4.5 AJUSTES NO MODELO

Para adequar o *MusicVAE* ao contexto deste estudo, foram necessárias algumas mudanças no código-fonte da sua implementação na biblioteca Magenta. Durante a análise para determinação do modelo, foi proposto que o melhor caminho para seguir seria basear-se naquele treinado para o plugin *Drumify*, mencionado no capítulo 4.2.

Visto isso, foram investigados as configurações de modelos pré-definidas presentes no *configs.py*, na pasta correspondente ao *MusicVAE* no projeto. Neste arquivo, existem variadas configurações do modelo, todas com características diferentes determinantes para trazer os resultados esperados por cada uma.

> [...] Uma configuração é a definição que contém todos os valores relacionados com a estrutura da rede e hiperparâmetros. Ela engloba totalmente o tipo de codificador e decodificador, tamanhos de camadas, hiperparâmetros comuns como taxa de aprendizagem e outros atributos opcionais que podem ser úteis em cada caso. Todas as configurações são encontradas juntas no mapa Config, cada uma delas mapeada para uma estrutura de chave e valor. (ROBERTS et al., 2018)

A configuração do modelo mais simples relacionado ao plugin é definido como *groovae_2bar_tap_fixed_velocity*. Ele converte 2 compassos de um padrão “tap” de velocidade constante em uma peça de bateria. É definido com um codificador bidirecional LSTM e um decodificador LSTM de batidas. Para hiperparâmetros, possui tamanho do lote; tamanho máximo da sequência em compassos; medida do vetor latente z; número de unidades por camada da RNN do codificador; número de unidades por camada da RNN do decodificador; peso máximo do custo Kullback–Leibler (KL) e bits para excluir da perda KL por dimensão. Utiliza também um conversor de dados para batidas, presente no arquivo

data.py e responsável por transfazer bidirecionalmente representações como batida (*hit*)/velocidade (*velocity*)/compensação (*offset*). Como Gillick (2019) pontua, a batida representa se houve ocorrência de determinada nota na sequência, e velocidade representa a força da batida de bateria. Compensação nesse caso se refere à distância relativa e em que direção se encontra o tempo de cada nota em relação à 16.ª nota mais próxima. (GILLICK, 2019)

Com estes parâmetros, atuações de bateria são representadas com triplas de batida/velocidade/compensação e cada *timestep* se refere a uma batida fixa em uma grade, com espaçamento padrão de 16 notas. Batidas que não se encaixarem no ritmo consideram o valor da compensação. As batidas são binários [0,1]. As velocidades são valores contínuos entre [0,1]. Compensações são valores contínuos entre [-0,5, 0,5], redimensionados para [-1, 1] para tensores.

Cada *timestep* contém uma representação de bateria, que deve ser uma de 9 categorias definidas por padrão em uma lista no arquivo *drums_encoder_decoder.py*. Com as categorias padrão, a entrada e a saída em um único *timestep* é de comprimento 9x3 = 27. Portanto, uma única medida de bateria em uma grade de 16 notas é uma matriz de forma (16, 27).

Para alcançar os objetivos deste estudo com esforço hábil, foi priorizada a reutilização das pré-definições presentes na biblioteca. Analisando o conversor de dados e entendendo como a técnica chamada de “squash” — transformar batidas em um “tap” constante — funciona, foi possível identificar potenciais mudanças que auxiliariam a nortear o projeto. Como dito anteriormente, cada timestep contém uma representação de bateria, que pode ser uma de 9 categorias definidas no projeto. Como este trabalho visa gerar melodias de piano, essas categorias foram adequadas para uma simples listagem de notas deste instrumento, variando das alturas 0 ao 127 na notação MIDI, como pode ser constatado na Figura 10.

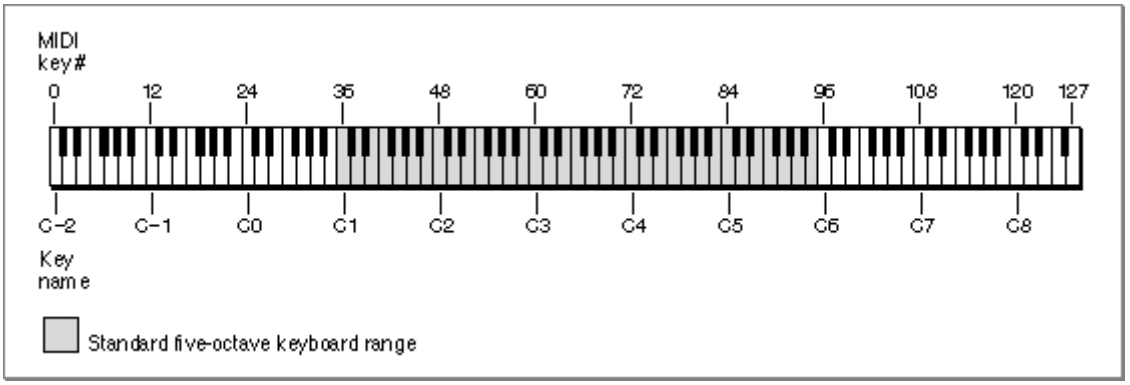


Figura 10 — Valores de notas MIDI e teclas de piano correspondentes
(APPLE COMPUTER INC, 1996)

Ademais, ao converter de volta dos tensores e iterar através dos instrumentos para criar uma nota da sequência, as informações da nota tiveram seu instrumento MIDI alterado de 9 para 1, ou seja, de bateria para piano. A propriedade *is_drum* foi definida como falsa, que determina se a nota é de uma bateria ou não.

4.6 TREINAMENTO

Após os ajustes compreendidos e aplicados, uma nova configuração foi criada e treinada para este estudo.

Para treinar um modelo com dados próprios, primeiro é necessário converter um conjunto de dados MIDI em um TFRecord de NoteSequences, ou seja, um arquivo compatível com variações do TensorFlow para armazenar uma sequência de registros binários, representando numericamente as notas musicais. Esta representação de dados armazena aspectos fundamentais das sequências de anotações (tempo, alturas, instrumentos, etc.) assim como metadados adicionais (etiquetas de seção, informações de acorde, etc.). É serializável e pode ser acessada e modificada através de várias linguagens. (ROBERTS, HAWTHORNE e SIMON, 2018)

Após isso, com o modelo ajustado na biblioteca local e o conjunto de dados pronto para treinamento, foram executados os comandos necessários para iniciar. Os modelos foram treinados para testagem, com hiperparâmetros adaptados às especificidades da máquina física usada no estudo, que possui limitações em aspectos como espaço, memória e GPU, como

visto no capítulo 4.1. O tamanho do lote foi diminuído de 512 para 32 e a taxa de aprendizado de 0,001 para 0,005. Cada modelo passou entre 3 e 12 horas em treino.

4.7 AVALIAÇÃO

Nesta etapa, foi executada uma análise do modelo considerando as métricas do capítulo 2.5 e outras com fins mais generalistas, visando verificar e potencialmente aprimorar o desempenho, a eficácia e a eficiência. Também foram utilizadas algumas ferramentas para isso, como o Tensorboard e o MGEval.

4.7.1 Tensorboard

O Tensorboard reúne ferramentas para visualização e experimentação de processos utilizados no TensorFlow. Dentro das suas capacidades, existem ações como rastreamento e visualização de métricas, exibição de dados, visualização de histogramas de parâmetros, conforme levantado por Rué Vilà (2020).

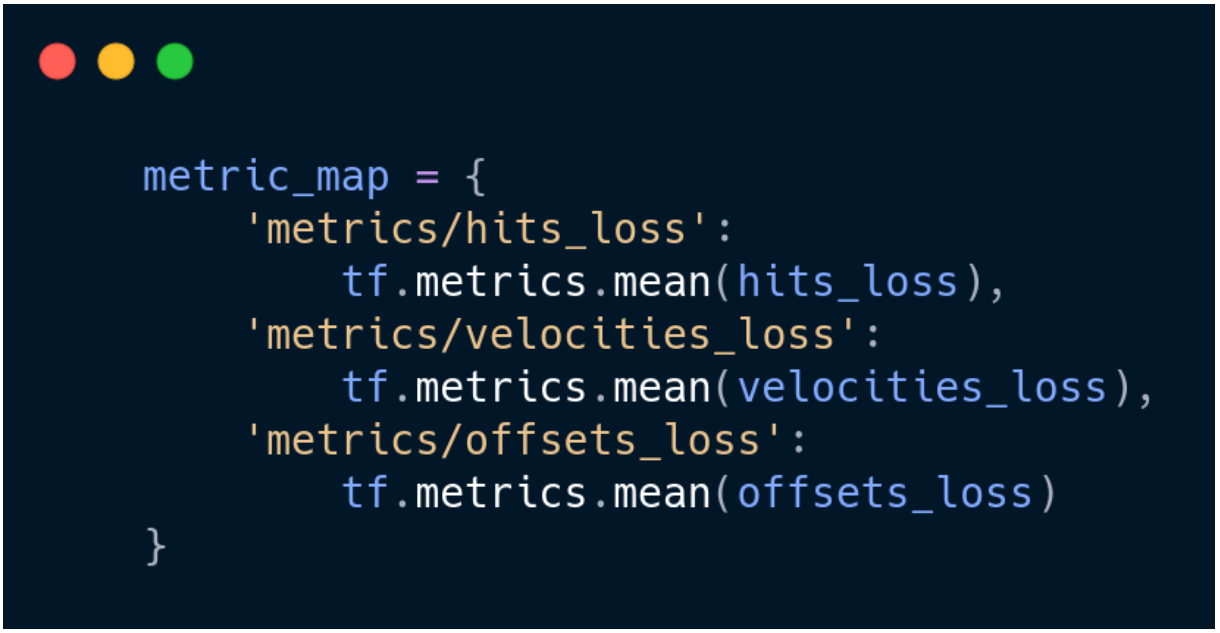
Ao longo do treinamento, o modelo salva rotineiramente um *checkpoint*, ou ponto de verificação, que guarda informações do estado atual dos parâmetros. Finalizando o treinamento, é possível visualizar a evolução em qualquer navegador com o uso do Tensorboard.

O código do modelo possui elementos do Tensorboard definidos, porém ignora o mapa métrico que corresponde a cada decodificador específico, ou seja, está utilizando uma métrica generalizada, e não parece ser o melhor rastreamento para todos os casos. O modelo está sempre utilizando o resumo geral para perdas (RUÉ VILÀ, 2020).

```
    scalars_to_summarize = {
        'loss': self.loss,
        'losses/r_loss': r_loss,
        'losses/kl_loss': kl_cost,
        'losses/kl_bits': kl_div / tf.math.log(2.0),
        'losses/kl_beta': beta,
    }
    metric_map = {
        'metrics/hits_loss':
            tf.metrics.mean(hits_loss),
        'metrics/velocities_loss':
            tf.metrics.mean(velocities_loss),
        'metrics/offsets_loss':
            tf.metrics.mean(offsets_loss)
    }
```

Figura 11 — Escalares para visualizar o progresso rastreado no navegador, extraído do script base_model.py

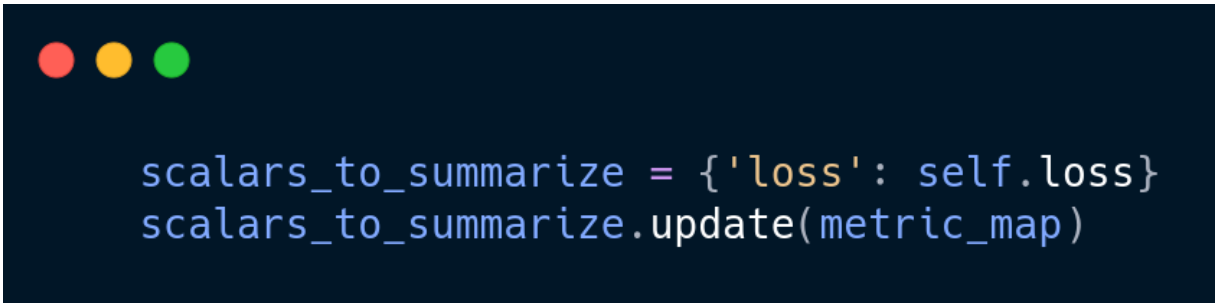
Para o modelo GrooVAE, o mapa métrico é definido em um método chamado *flat_reconstruction_loss* no GrooveLSTMDecoder. Observando a Figura 11, é possível identificar que a perda de batidas, velocidades e compensações são estimadas em separado e e reunidas no mapa métrico. Ademais, são estimados conjuntamente no atributo de perda do objeto modelo.



```
metric_map = {
    'metrics/hits_loss':
        tf.metrics.mean(hits_loss),
    'metrics/velocities_loss':
        tf.metrics.mean(velocities_loss),
    'metrics/offsets_loss':
        tf.metrics.mean(offsets_loss)
}
```

Figura 12 — Mapa métrico para o modelo GrooVAE, extraído de lstm_models.py

De todo modo, no modelo padrão, esses valores são ignorados e os escalares gerais são usados. Para contornar esse caso, foi implementado o código da Figura 13 para substituir aquele apresentado na Figura 11, assim como foi aplicado por Rué Vilà (2020).



```
scalars_to_summarize = {'loss': self.loss}
scalars_to_summarize.update(metric_map)
```

Figura 13 — Atualização de escalares para se adequar à métrica do GrooVAE necessária para a visualização, editada no base_model.py

A visualização obtida com o Tensorboard é similar às apresentadas nas Figuras 14 e 15:

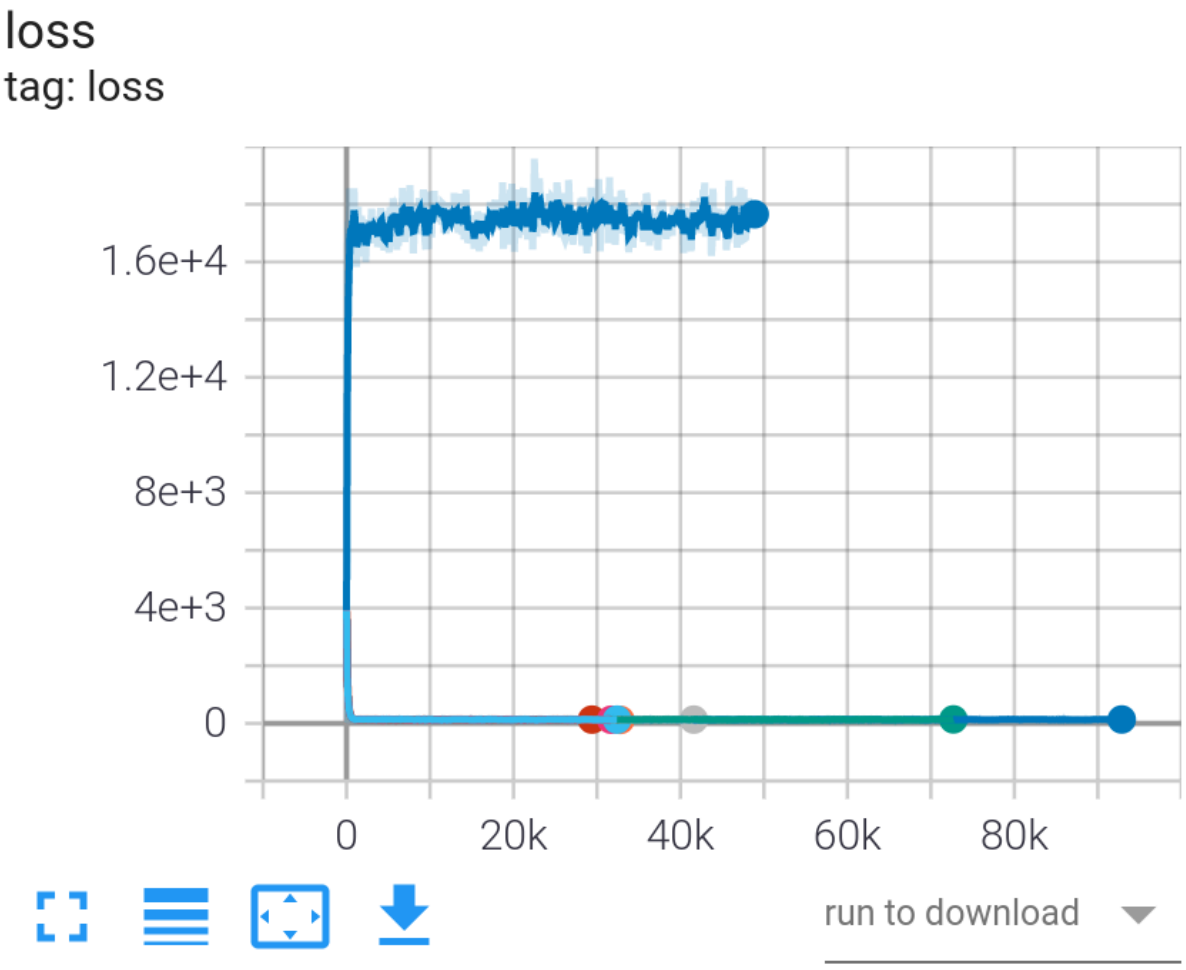


Figura 14 — Gráfico de perdas geral

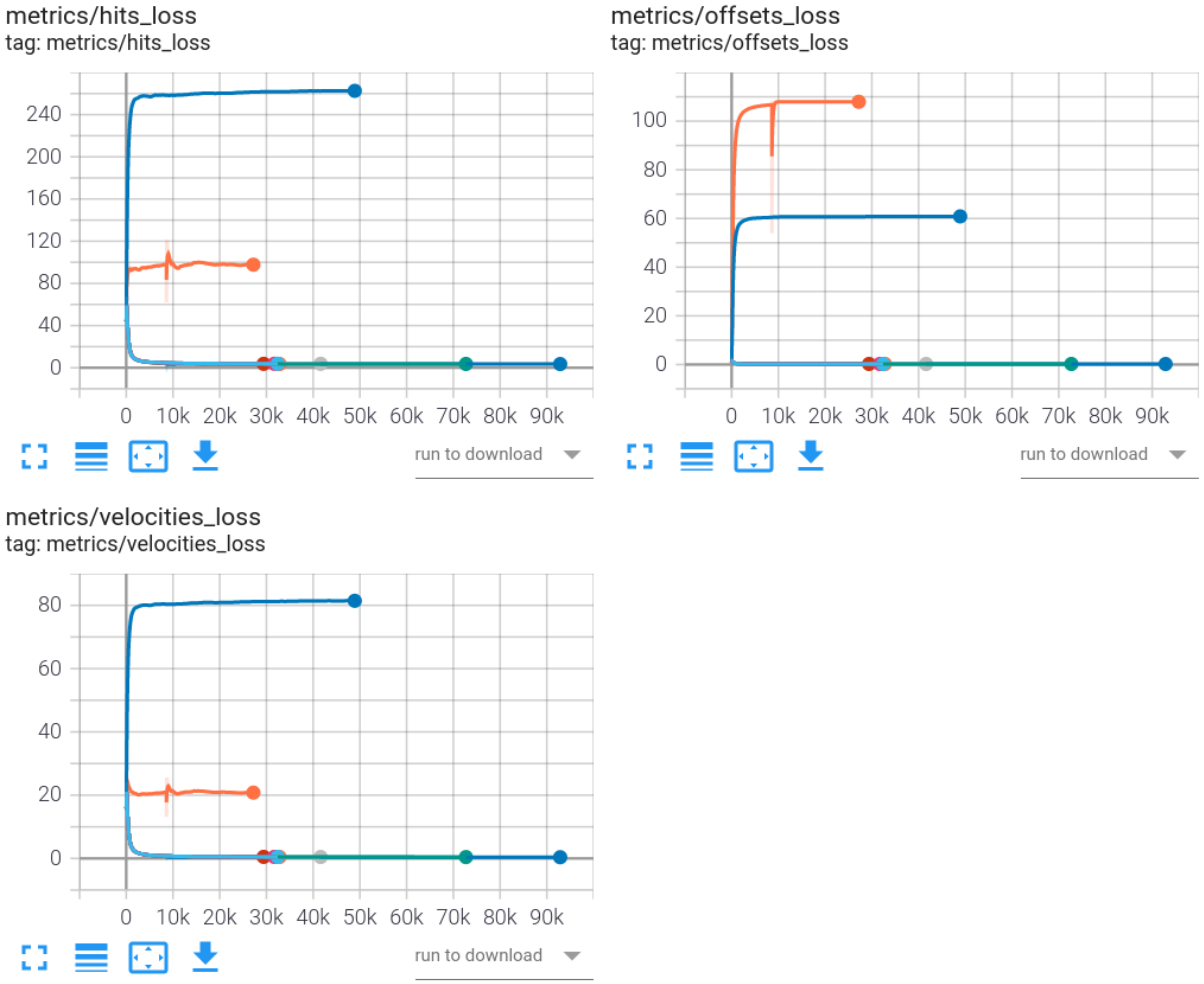


Figura 15 — Ampliação das métricas que acompanham as perdas de batidas, compensações e velocidades

Na Figura 14, é possível visualizar a perda global, ou seja, a soma das três perdas específicas. Já na área inferior, estão dispostas as perdas específicas. No eixo y, é identificável a quantia da perda calculada, e o eixo x representa o passo no otimizador. Sobre a perda, vemos uma faixa diferente na perda de batidas do que nas outras duas. Isto porque a perda de batidas é uma perda de entropia cruzada, as compensações e velocidades são erros quadráticos médios (RUÉ VILÀ, 2020). Dado o contexto geracional do estudo, aparenta-se que uma perda com erro quadrático médio seria mais otimizada que a entropia cruzada, visando esse valor para melhoria na perda de batidas.

```python
def _flat_reconstruction_loss(self, flat_x_target, flat_rnn_output):
  # flat_x_target is by default shape (1,27), [on/offs... vels...offsets...]
  # split into 3 equal length vectors
  target_hits, target_velocities, target_offsets = tf.split(
      flat_x_target, 3, axis=1)

  output_hits, output_velocities, output_offsets = self._activate_outputs(
      flat_rnn_output)

  hits_loss = tf.reduce_sum(tf.losses.log_loss(
      labels=target_hits, predictions=output_hits,
      reduction=tf.losses.Reduction.NONE), axis=1)

  velocities_loss = tf.reduce_sum(tf.losses.mean_squared_error(
      target_velocities, output_velocities,
      reduction=tf.losses.Reduction.NONE), axis=1)

  offsets_loss = tf.reduce_sum(tf.losses.mean_squared_error(
      target_offsets, output_offsets,
      reduction=tf.losses.Reduction.NONE), axis=1)

  loss = hits_loss + velocities_loss + offsets_loss

  metric_map = {
      'metrics/hits_loss':
          tf.metrics.mean(hits_loss),
      'metrics/velocities_loss':
          tf.metrics.mean(velocities_loss),
      'metrics/offsets_loss':
          tf.metrics.mean(offsets_loss)
  }

  return loss, metric_map
```

Figura 16 — Código do método _flat_reconstruction_loss extraído de lstm_models.py

Observando a Figura 16, pode-se visualizar o código do método *_flat_reconstruction_loss* no GrooveLSTMDecoder, contendo as operações necessárias para o cálculo das perdas.

4.7.2 MGEval

MGEval, assim como o Tensorboard, é uma caixa de ferramentas de código aberto. Ela dispõe de métodos de avaliação, análise e visualização para sistemas musicais generativos, conforme pontuado por Yang e Lerch (2020). Seu objetivo é fornecer um sistema

de avaliação objetivo para ter uma melhor confiabilidade, validade e reprodutibilidade na geração musical.

MGEval é responsável por calcular as métricas descritas no capítulo 2.5 e outras adicionais, propondo um conjunto de medidas objetivas musicalmente informadas para obter análises que impulsionam ainda mais séries de estratégias de avaliação.

Como também visto anteriormente, as métricas são categorizadas como absolutas ou relativas. Respectivamente, fornecendo intuições sobre as propriedades do conjunto de dados gerado ou coletado, e comprando dois conjuntos de dados de treinamento e gerado.

A estrutura do código é bastante simples, presente em sua maioria no arquivo *core.py*. Nele, existe uma função para extração de *features* de determinado arquivo MIDI utilizando outras ferramentas e uma classe com as funções responsáveis por calcular as características de fato. As complexas funções utilizam como base os objetos montados pelas bibliotecas *pretty_midi* e *python-midi*, que fornecem diversas informações sobre as faixas fornecidas.

Adicionalmente, existe o arquivo *utils.py*, que armazena as funções responsáveis pelo cálculo de distâncias das métricas avaliativas, KLD e OA. Boa parte utiliza as bibliotecas *sklearn*, *numpy* e *scipy* para o manejo de dados e contas melindrosas.

Conta também com o arquivo *__main__.py*, o principal do projeto, responsável por chamar as demais funcionalidades do pacote corretamente. Dessa forma, recebe os argumentos definidos pelo usuário, incluindo os diretórios com os arquivos MIDI gerados e de teste, o nome do arquivo de saída e o número de compassos a serem considerados nas faixas; com isso, inicia e gerencia a extração de *features* e cálculo de métricas, bem como salva as informações em um arquivo JavaScript Object Notation (JSON). JSON é um formato de texto para serialização de dados estruturados. É derivado do objeto literal do JavaScript. (CROCKFORD, 2006)

Apesar da biblioteca possuir todo esse ecossistema, ainda precisaram ser feitos alguns ajustes para o funcionamento pleno, especialmente no Python 3. A atualização de algumas dessas bibliotecas, como a *python-midi*, foi realizada, bem como codificação extra para contornos de casos específicos, cálculos e gráficos adicionais. Bons exemplos destes cálculos

são a distância de Wasserstein entre o histograma da classe de altura dos arquivos MIDI gerados e de teste, e entre o histograma de comprimento de nota dos mesmos arquivos. Os gráficos apresentam a medição da diferença de distâncias *intra-set* e *inter-set* por KLD e OA.

O comando executado para executar a avaliação é apresentado na Figura 17.

```
python {$PATH_TO_MGEVAL} \
--set1dir={$PATH_TO_EVAL_MIDI} \
--set2dir={$PATH_TO_GENERATED_MIDI} \
--outfile={$OUTPUT_FILENAME} \
--num-bar={$NUMBER_OF_BARS}
```

Figura 17 — Código executado para avaliar os arquivos MIDI

Com isso, o arquivo *__main__.py* é executado e com ele todas as funcionalidades da MGEval. Após a finalização do processo, os resultados podem ser visualizados pelo terminal ou, de uma forma mais estruturada, em um JSON com as informações como evidenciado no Apêndice A.

Conforme mencionado no capítulo 2.5, para cada *feature*, é calculada a média e o desvio padrão dos arquivos MIDI gerados e de teste, bem como duas métricas considerando as PDFs das distâncias *intra-set* do conjunto de treinamento e *inter-set* entre os conjuntos de dados de treinamento e gerado: OA e KLD. Adicionalmente, nas *features* que retornam um histograma, calcula a distância de Wasserstein entre a média dos arquivos MIDI gerados e de teste. A menor diferença, ou distância, sugere uma melhor imitação de estilo do conjunto de dados. (JIANG, 2020)

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
Este trabalho	+1.90	+1.13	+0.015	0.083	0.015	0.0001±0.016	0.27±0.01	0.83±0.02

Tabela 5 — Tabela com métricas baseadas nos arquivos gerados e de teste deste trabalho

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Este trabalho	3.15	7.71	0.13	-	-

Tabela 6 — Tabela com métricas do conjunto de dados

Para chegar nos valores apresentados nas Tabelas 5 e 6, foram realizados alguns cálculos com o resultado retornado pela execução da ferramenta. No que condiz aos valores do conjunto de dados, apresentados na Tabela 6, foram realizados da seguinte maneira: O *PC/Bar* foi adquirido realizando a média entre os dois valores apresentados na primeira posição apresentado como *bar_used_pitch* no arquivo JSON; O *PI* é simplesmente o primeiro valor retornado como *avg_pitch_shift;* O valor do *IOI* também é exatamente o mesmo retornado na primeira posição como *avg_IOI*; *PCH* e *NLH* são inexistentes para o conjunto de dados, visto que estes são cálculos de distância entre arquivos gerados e de teste.

Agora considerando as métricas do modelo, apresentados na Tabela 5, as representações são um pouco diferentes, sendo relatadas como as diferenças das métricas dos arquivos gerados em relação ao conjunto de dados de teste nas principais *features* e a distância de Wasserstein para *PCH* e *NLH*. Já para a distribuição da altura (D_P), duração (D_D) e intervalo entre notas (D_{IOI}), é apresentada a OA dos arquivos gerados, em conjunto com a divergência de Kullback-Leibler, também considerando os mesmos arquivos. Ou seja, o valor na sétima posição, em conjunto com o valor na oitava posição para *total_pitch_class_histogram*, *note_length_hist* e *avg_IOI*, respectivamente.

Para as primeiras 5 métricas, esperamos um valor mais próximo possível de 0, visto que demonstra a distância dos valores de referência. Em contraste, para as distribuições, esperamos um valor mais próximo possível de 1, considerando que representa a área

sobreposta entre arquivos de teste e aqueles gerados pelo modelo. Estes valores demonstram, acima de tudo, a similaridade entre os conjuntos de dados fornecidos para a ferramenta, conforme pontuado por Yang e Lerch (2020), neste trabalho sendo o conjunto de teste, definido no capítulo 4.2, e o de arquivos gerados.

4.8 OTIMIZAÇÃO DO MODELO

Primeiramente, foram realizados treinamentos com foco apenas no ajuste da taxa de aprendizagem do modelo. Esta taxa informa a grandiosidade das atualizações em cada etapa do otimizador.

O comando executado para dar início ao ciclo de treinamento pode ser visualizado na Figura 18.

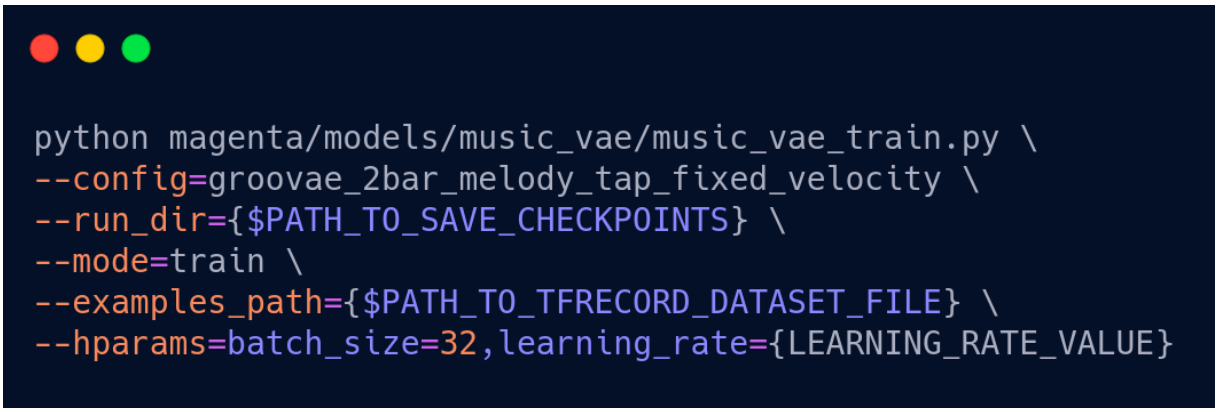


Figura 18 — Código executado para treinar o modelo

Um diretório pai é determinado para salvar os pontos de verificação de cada treinamento em pastas dedicadas, simplificando assim a visualização dos resultados usando o Tensorboard.

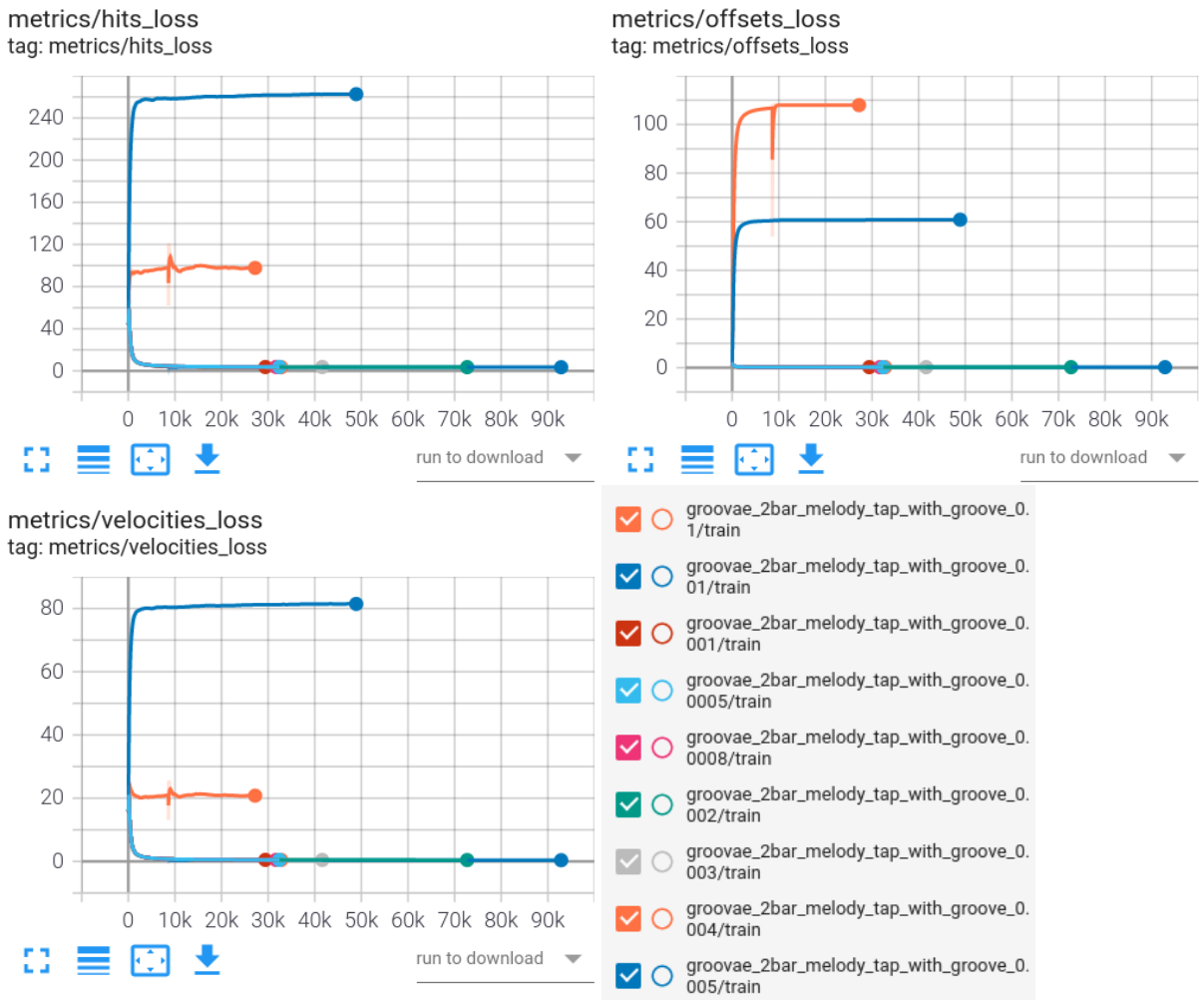
Foram testadas taxas de aprendizado com valor 0.1, 0.01, 0.001, 0.0005, 0.0008, 0.002, 0.003, 0.004 e 0.005, assim como o trabalho de Rué Vilà (2020). Finalizando o treinamento para cada taxa, é possível visualizar e comparar os modelos pelo navegador. Essa

comparação torna possível a determinação do valor da taxa de aprendizagem que gera o menor custo.

Para observar os resultados no navegador através do Tensorboard, o comando da Figura 19 deve ser executado, selecionando o diretório pai com os pontos de verificação.

```
tensorboard --logdir=$FOLDER_NAME
```

Figura 19 — Código visualizar os gráficos no navegador com Tensorboard



Figuras 20 e 21 — Gráficos de perda e Legenda de cores

Name	Smoothed	Value	Step	Time	Relative
groovae_2bar_melody_tap_with_groove_0.0005/train	3.883	3.881	26.9k	Thu Jul 14, 17:38:18	3h 34m 51s
groovae_2bar_melody_tap_with_groove_0.0008/train	3.858	3.856	26.9k	Thu Jul 14, 22:13:08	3h 40m 37s
groovae_2bar_melody_tap_with_groove_0.001/train	3.749	3.748	26.9k	Thu Jul 14, 13:37:36	3h 58m 2s
groovae_2bar_melody_tap_with_groove_0.002/train	3.783	3.782	26.9k	Fri Jul 15, 02:59:51	4h 3m 48s
groovae_2bar_melody_tap_with_groove_0.003/train	3.823	3.821	26.9k	Fri Jul 15, 13:58:04	4h 0m 53s
groovae_2bar_melody_tap_with_groove_0.004/train	3.645	3.643	26.9k	Fri Jul 15, 19:50:52	3h 32m 49s
groovae_2bar_melody_tap_with_groove_0.005/train	3.497	3.497	26.86k	Sat Jul 16, 04:17:57	7h 35m 46s
groovae_2bar_melody_tap_with_groove_0.01/train	261.4	261.5	26.9k	Thu Jul 14, 06:09:17	4h 24m 28s
groovae_2bar_melody_tap_with_groove_0.1/train	97.74	97.7	26.91k	Wed Jul 13, 22:52:18	4h 25m 54s

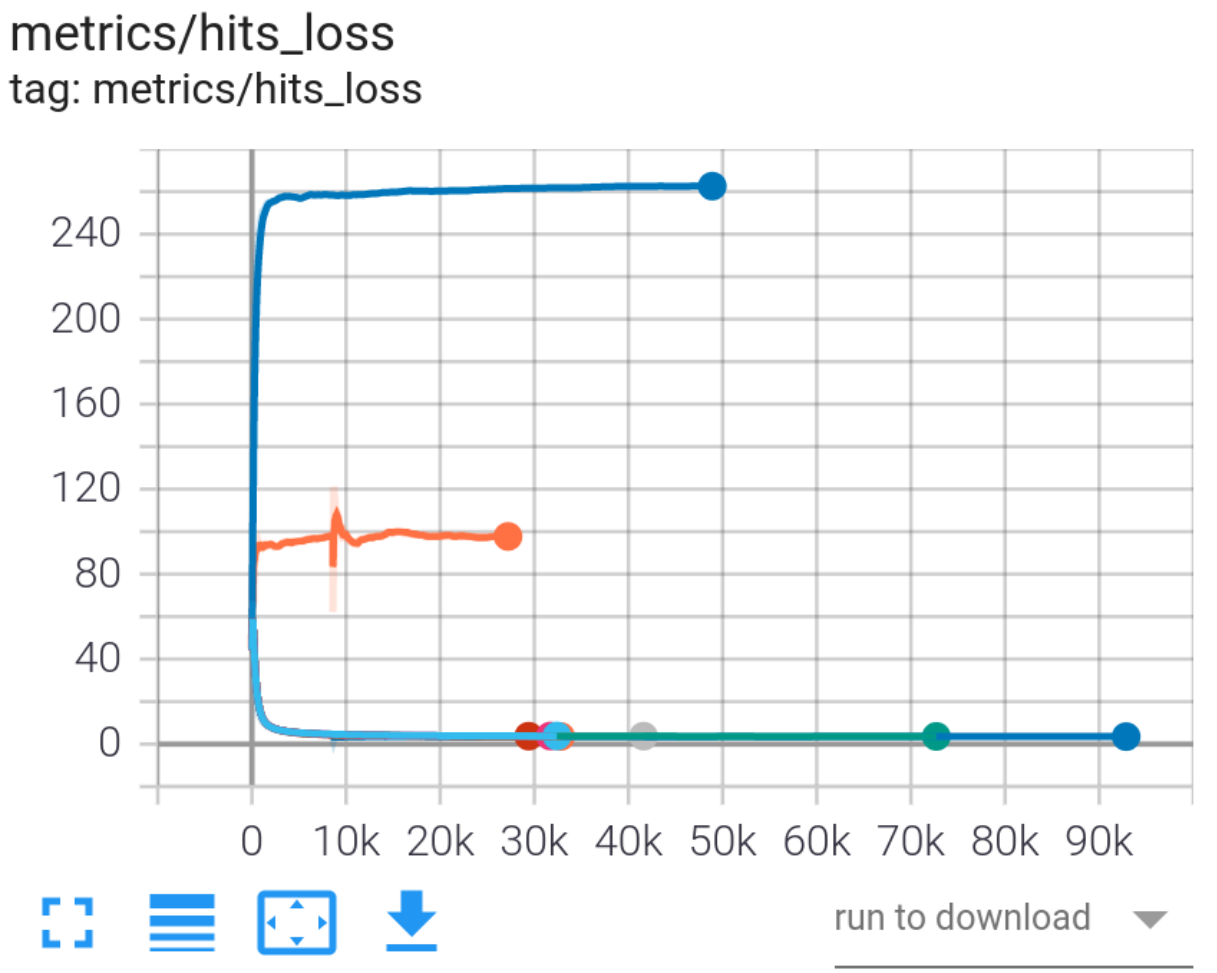


Figura 22 — Ampliação da perda de batidas

Name	Smoothed	Value	Step	Time	Relative
groovae_2bar_melody_tap_with_groove_0.0005/train	0.2168	0.2166	26.9k	Thu Jul 14, 17:38:18	3h 34m 51s
groovae_2bar_melody_tap_with_groove_0.0008/train	0.2183	0.2181	26.9k	Thu Jul 14, 22:13:08	3h 40m 37s
groovae_2bar_melody_tap_with_groove_0.001/train	0.2312	0.231	26.9k	Thu Jul 14, 13:37:36	3h 58m 2s
groovae_2bar_melody_tap_with_groove_0.002/train	0.2246	0.2243	26.9k	Fri Jul 15, 02:59:51	4h 3m 48s
groovae_2bar_melody_tap_with_groove_0.003/train	0.2306	0.2304	26.9k	Fri Jul 15, 13:58:04	4h 0m 53s
groovae_2bar_melody_tap_with_groove_0.004/train	0.2297	0.2295	26.9k	Fri Jul 15, 19:50:52	3h 32m 49s
groovae_2bar_melody_tap_with_groove_0.005/train	0.2171	0.217	26.86k	Sat Jul 16, 04:17:57	7h 35m 46s
groovae_2bar_melody_tap_with_groove_0.01/train	60.76	60.76	26.9k	Thu Jul 14, 06:09:17	4h 24m 28s
groovae_2bar_melody_tap_with_groove_0.1/train	108	108	26.91k	Wed Jul 13, 22:52:18	4h 25m 54s

metrics/offsets_loss metrics/velocities_loss

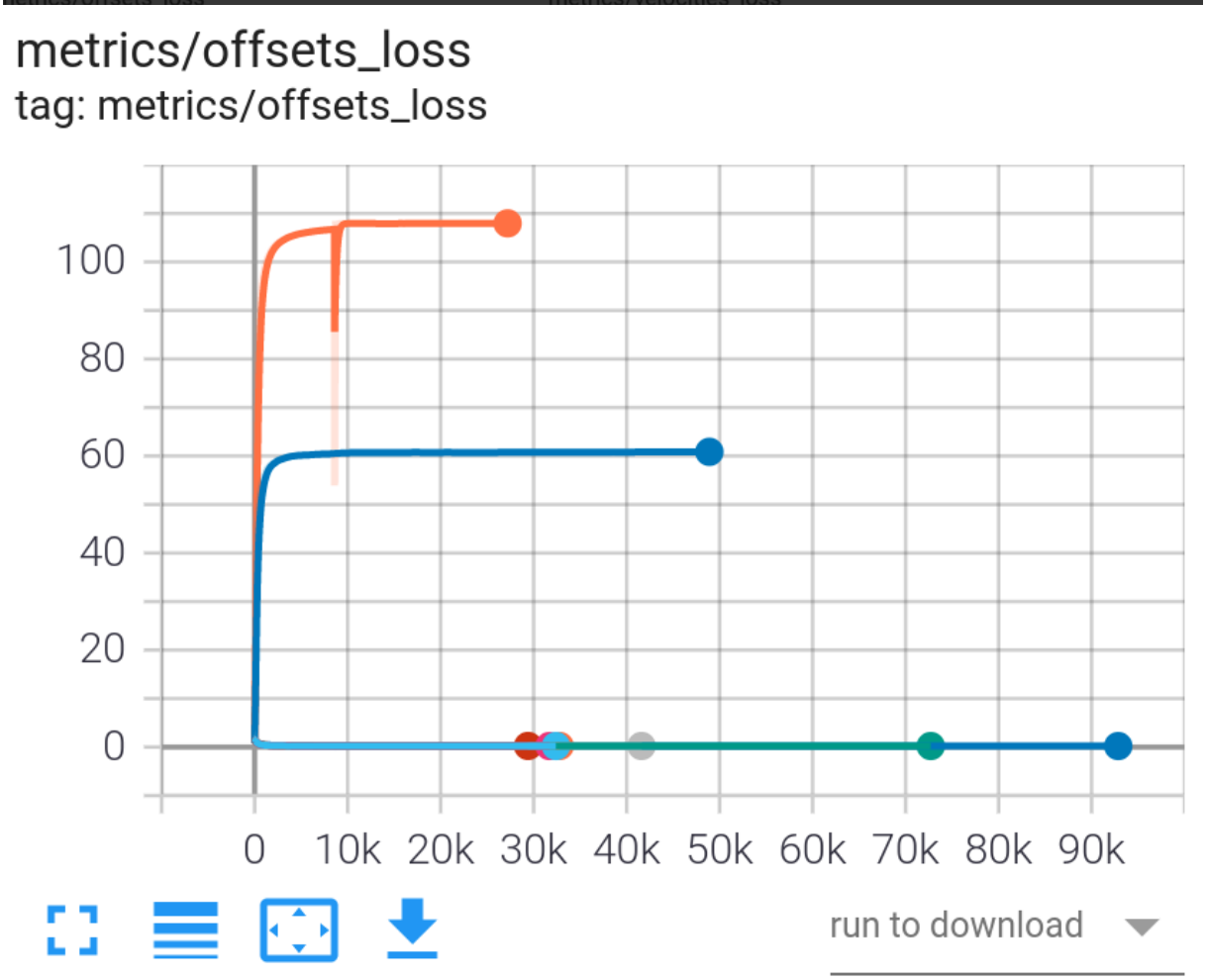


Figura 23 — Ampliação da perda de compensações

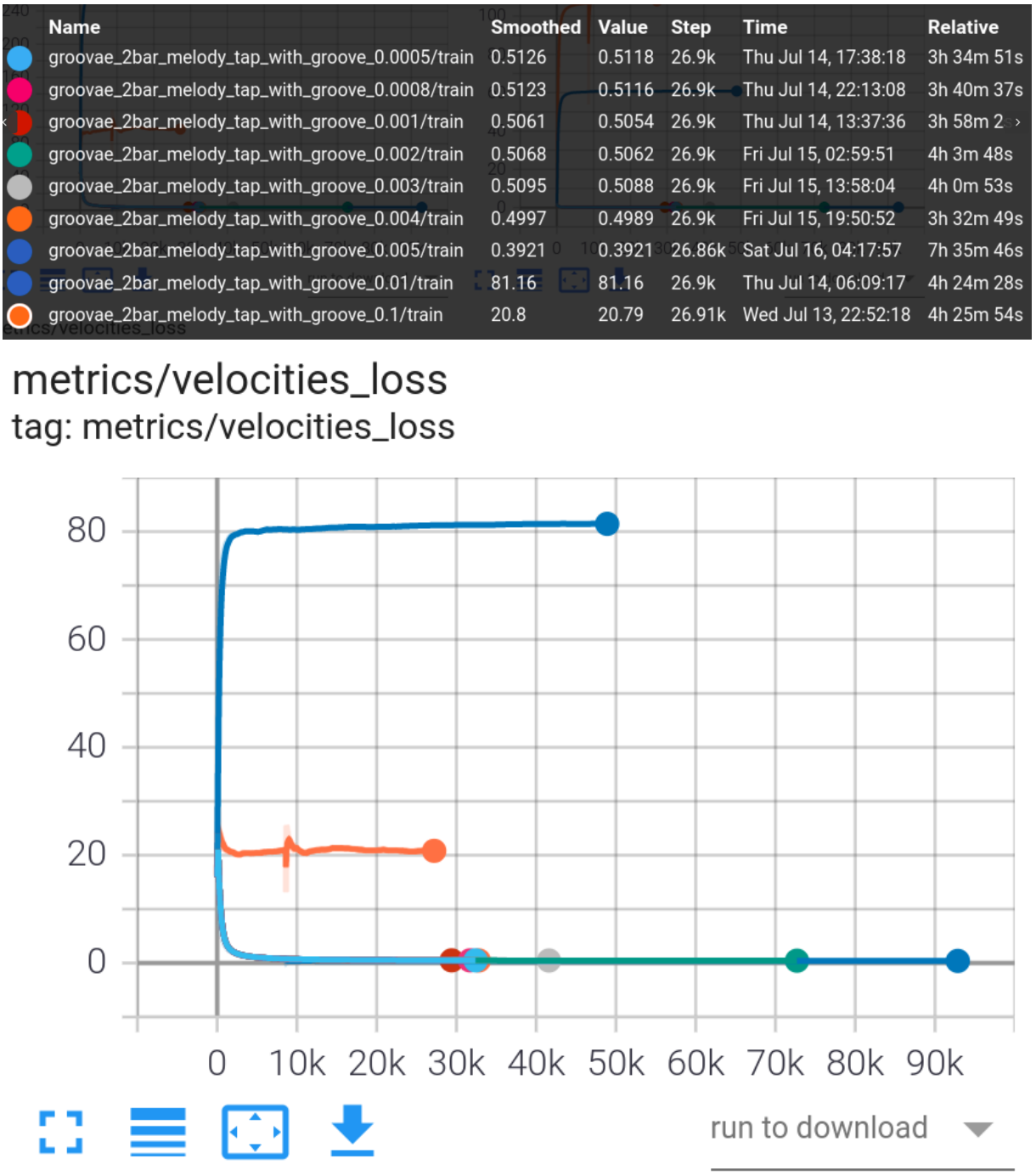


Figura 24 — Ampliação da perda de velocidade

A Figura 20 mostra os principais gráficos de perda obtidos utilizando o Tensorboard. Analisando e ranqueando os dados presentes nas Figuras 22, 23 e 24, podemos concluir que, de maneira geral, a taxa de aprendizado com valor 0.005 apresentou os resultados mais equilibrados e satisfatórios entre os três tipos de perda.

Na sequência, foram comparados diversos tamanhos de rede. O valor padrão vem definido na seguinte estrutura:

- enc_rnn = [512]
- dec_rnn = [256,256]
- z_size = 256

Ainda seguindo os experimentos de Rué Vilà (2020), manteve-se a estrutura de enc_rnn=[X], dec_rnn=[X/2,X/2], z_size=X/2, e testaram-se os seguintes valores para X: 256, 384, 512, 768 e 1024. Com a taxa de aprendizagem fixada em 0,005, o código presente na Figura 25 foi executado.

```
python magenta/models/music_vae/music_vae_train.py \
--config=groovae_2bar_melody_tap \
--run_dir={$PATH_TO_SAVE_CHECKPOINTS} \
--mode=train \
--examples_path={$PATH_TO_TFRECORD_DATASET_FILE} \
--hparams=batch_size=32,learning_rate=0.005,enc_rnn_size="[X]",dec_rnn_size="[X/2,X/2]",z_size=X/2
```

Figura 25 — Código para treinamento com tamanhos de rede customizados

Utilizando o comando da Figura 19, é possível visualizar os resultados destes treinamentos.

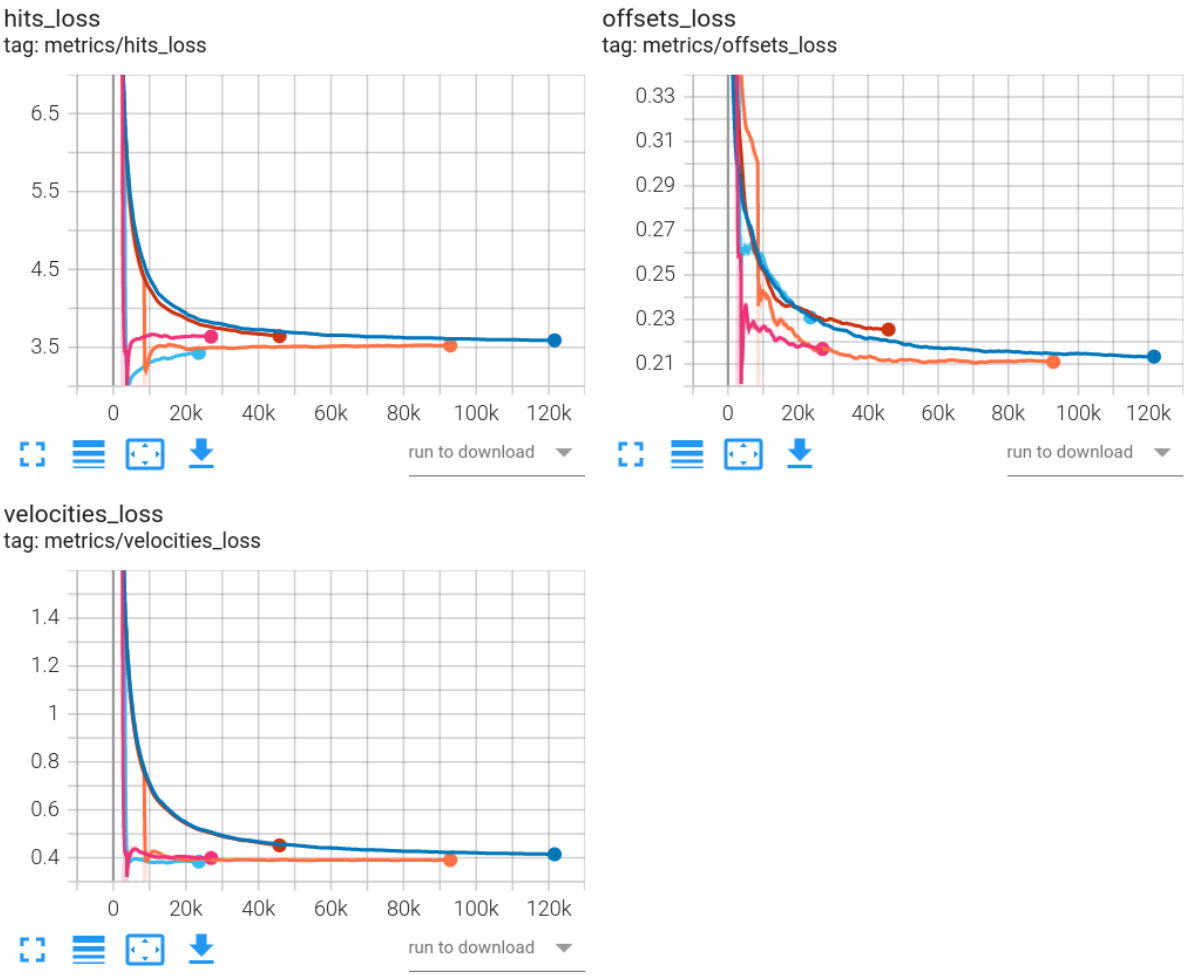


Figura 26 — Comparação entre os diferentes tamanhos das camadas de rede

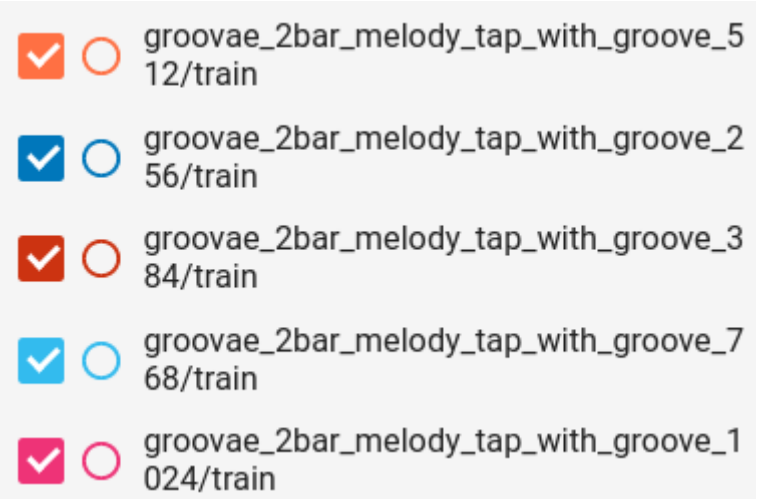


Figura 27 — Legenda

Tamanho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
256	+2.40	+0.51	-0.025	0.083	0.015	0.0003±0.35	0.27±0.01	0.63±0.06
384	+2.15	**+0.31**	**-0.009**	0.083	0.015	0.0001±0.08	0.27±0.01	0.73±0.07
512	+3.30	+1.09	-0.021	0.083	0.015	**0.015±0.14**	0.27±0.01	0.69±0.04
768	**+1.90**	+1.13	+0.015	0.083	0.015	0.0001±0.016	0.27±0.01	**0.83±0.02**
1024	+2.45	+1.96	-0.033	0.083	0.015	0.0018±0.078	0.27±0.01	0.53±0.04

Tabela 7 — Tabela com métricas dos modelos com diferentes tamanhos de rede

Com o comando da Figura 17, podemos avaliar os arquivos gerados pelos modelos treinados com diferentes tamanhos de rede, similar ao que foi realizado no capítulo 4.7.2. Os resultados podem ser analisados na Tabela 7.

Como pode ser concluído através da Figura 26 e da Tabela 7, o codificador que teve o melhor desempenho foi o de tamanho de rede = [768], decodificador = [384,384] e z_size = 384.

4.9 DESENVOLVIMENTO DO PROTÓTIPO

A etapa final foi executada visando alcançar um dos principais objetivos propostos neste estudo, implementar um protótipo com interface para tornar hábil e conveniente o uso do modelo em seu contexto de plugin, assim como o *Drumify*, mencionado no capítulo 4.2. Aqui, um servidor simples com Node.js foi implementado para servir os arquivos estáticos do modelo. A parte do cliente foi desenvolvida com Vue.js na versão 3 como arcabouço principal, bem como Stylus para estilização, Magenta.js (ROBERTS, HAWTHORNE e SIMON, 2018) para consumir e aplicar o modelo, e outras bibliotecas ou pacotes. Esta etapa teve grande foco no visual e no interativo, o que considera a preocupação com a usabilidade da aplicação visando aprimorar a experiência do usuário final. Também houve grande

inspiração dos plugins já existentes da biblioteca Magenta, como o próprio *Drumify*. Desta forma, a versão deste estudo foi nomeada *Melodify*.

Antes da implementação dos projetos, foi necessário converter os pesos dos *checkpoints* do modelo treinado para o formato utilizado pela Magenta.js, diferente daquele fornecido como saída dos treinamentos da biblioteca python Magenta, utilizando o script *checkpoint_converter*, disponibilizado na versão JavaScript (JS) da biblioteca.

Além disso, também foi preciso especificar determinados parâmetros do modelo em um arquivo JSON, especificado como *config.json* no mesmo diretório que os *checkpoints* convertidos. Este arquivo de configuração contém todas as informações necessárias (além dos pesos) para instanciar e executar o modelo: a categoria de modelo e a especificação do conversor de dados mais a codificação de acordes opcional, entradas auxiliares, entre outras opções.

Seguindo, foi implementado o servidor responsável por servir os arquivos convertidos e criados nos passos anteriores para a aplicação no lado do cliente. As principais tecnologias utilizadas foram Node.js e Express, arcabouço para rodar JS como *back-end* e um gerenciador de rotas para Node.js, respectivamente. Outros pacotes também foram utilizados, como um encarregado de manejar as requisições e prevenir erros *Cross-Origin*, ou seja, suporta que materiais privados sejam recuperados por um domínio diferente do domínio ao qual pertence o material que será recuperado. Esta aplicação serve na porta 3000 da máquina local e possui apenas a rota */static*, capaz de retornar os arquivos dos *checkpoints* convertidos do modelo, que devem estar presentes dentro de */models/converted* no diretório do projeto. O código-fonte pode ser observado na Figura 28.

```
const express = require('express');
const cors = require('cors');
const app = express();
const port = 3000;

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use('/static', express.static(__dirname + '/models/converted'));


app.listen(port, () => console.log(`CORS-enabled web server is listening to port ${port}`));
```

Figura 28 — Código-fonte do servidor do Melodify

Como mencionado anteriormente, houve grande inspiração nos plugins já existentes na Magenta Studio (ROBERTS et al., 2019). A interface apresentada simplifica ao máximo o uso de um modelo que grande parte dos potenciais usuários alvo, como músicos e afins, poderiam considerar complexo demais de outra forma, já que não necessariamente possuem grande aptidão tecnológica. Desta forma, é possível apenas enviar um arquivo MIDI de qualquer instrumento e receber uma atuação de bateria para acompanhamento. Porém, diferentemente do protótipo apresentado neste trabalho, os plugins da Magenta Studio são maduros e robustos, além de utilizarem React ao invés de Vue como arcabouço principal e Electron, para rodarem como aplicações nativas desktop ao invés de diretamente no browser. O plugin *Drumify* pode ser visualizado na Figura 29.

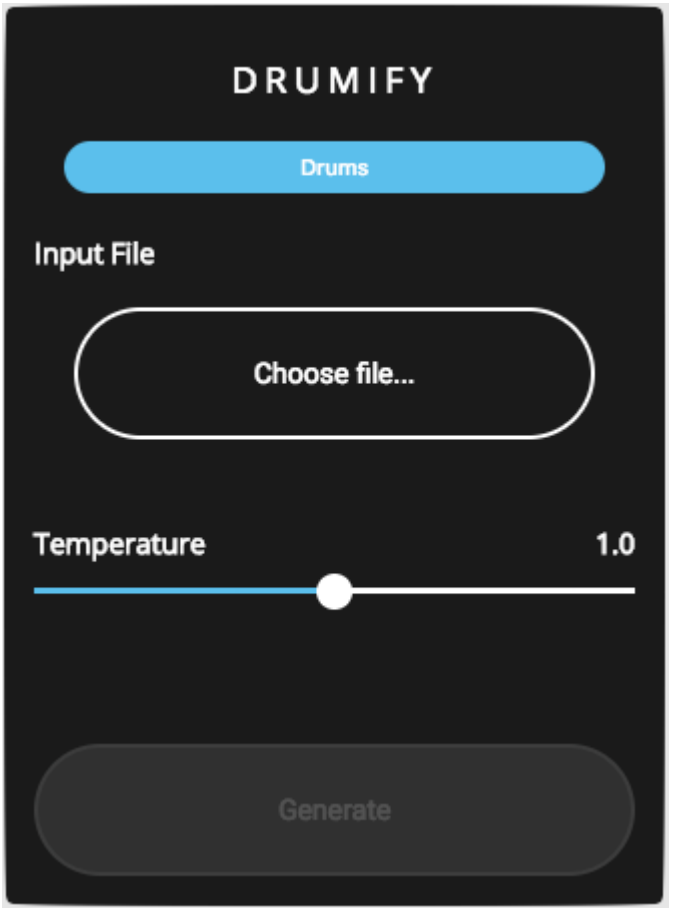


Figura 29— Interface do Drumify versão standalone

Dessa forma, foi criada a parte do cliente do *Melodify*. O protótipo utiliza Vue na versão 3, Stylus e Magenta.js (ROBERTS, HAWTHORNE e SIMON, 2018). Primeiramente, foi criada uma classe para utilizar a Magenta.js e consumir o modelo que será servido através do servidor descrito acima. Essa classe também é responsável por receber a entrada de bateria em formato MIDI enviada pelo usuário e, posteriormente, gerar a melodia para acompanhamento. Além da classe, foram criados componentes de botão, entrada de arquivo, rótulo e seleção em intervalo. A tela principal é implementada com ditos componentes.

Após enviar o arquivo MIDI contendo a entrada com uma atuação de bateria e selecionando a temperatura, o botão *Generate* fica habilitado e o usuário pode clicar para gerar suas melodias de acompanhamento. O protótipo está apresentado na Figura 30.

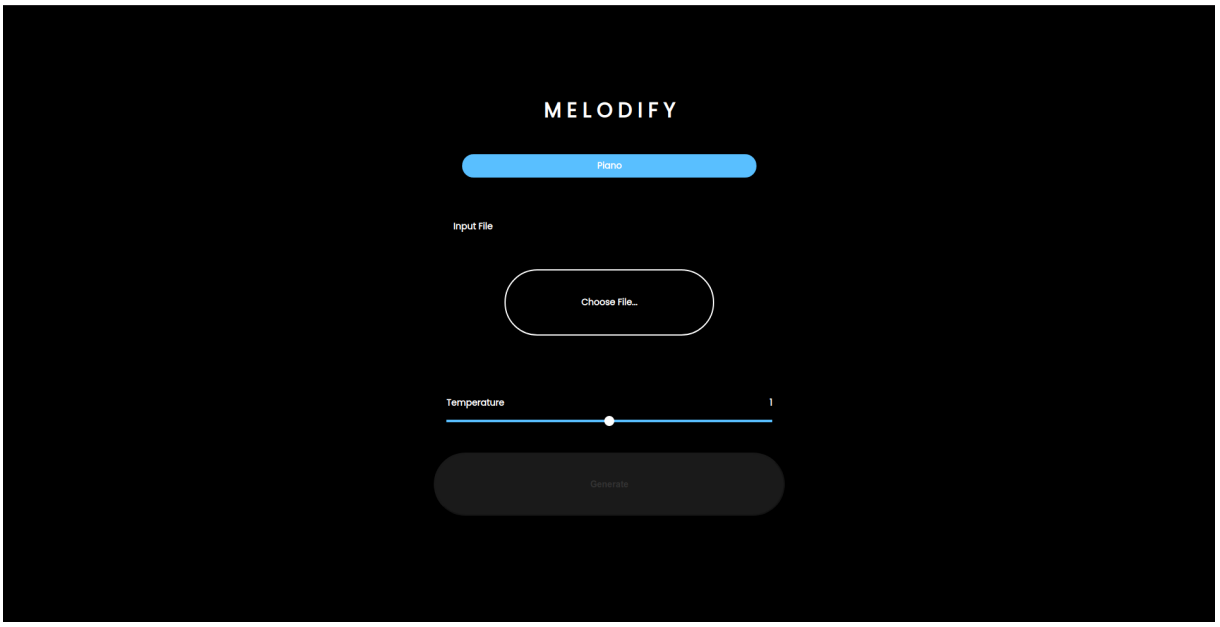


Figura 30 — Interface do Melodify rodando no browser

4.10 COMPARAÇÃO COM OS TRABALHOS RELACIONADOS

Comparando com os trabalhos relacionados, é perceptível que, similarmente ao trabalho 1, de Jiang et al. (2020) (1) e 2, de Ren et al. (2020) (2), neste trabalho foi usado um conjunto de dados existente, ou seja, não executou um processo de coleta de dados, como visto previamente no capítulo 4.2 e apresentado na Tabela 8.

Uma diferença notável fica por parte da técnica de ML. O trabalho 1 utiliza aprendizado por reforço, enquanto o segundo adota o aprendizado supervisionado, e este aplica aprendizado não supervisionado. Outra diferença é a avaliação. O trabalho 1 e 2 recorreram a técnicas objetivas e subjetivas para avaliar seus resultados, enquanto este efetuou uma avaliação majoritariamente objetiva. O trabalho 1 utilizou métricas objetivas que também foram utilizadas neste projeto, mas também aplicou um extenso estudo com usuários, que considerava diferentes níveis de conhecimento musical ao avaliar os resultados. O trabalho 2 também aplicou métricas que estão aqui apresentadas, além de um estudo humano com 15 participantes, tendo 5 que compreendem a teoria básica da música. Este trabalho, por conta de tempo hábil e escopo, tem uma avaliação majoritariamente objetiva conforme visto no capítulo 4.7. A parte subjetiva pode ser dada pelas opiniões pessoais.

Trabalho (n.º)	Técnica de ML	Conj. de dados	Avaliação
RL-Duet (1)	Apr. por reforço	Bach Chorale	Obj. e Subj.
PopMAG (2)**	Apr. supervisionado	LMD*, CPMD e FreeMidi	Obj. e Subj.
Este trabalho	Apr. não supervisionado	Groove MIDI Dataset	Majoritariamente Obj.

Tabela 8 — Tabela comparativa das técnicas, conjuntos de dados e categoria de avaliação entre os artigos coletados e este trabalho

Levando em conta as métricas, podem ser comparadas e estão dispostas na Tabela 9. O *PC/Bar* apresentou uma diferença de +1.90 em relação ao conjunto de dados de teste; enquanto o *PI* e o *IOI* mostraram diferenças de +1.13 e +0.015, respectivamente. As distâncias de Wasserstein entre a música gerada e o conjunto de dados de teste para *PCH* e *NLH* foram definidas como 0.083 e 0.015, também respectivamente. As distribuições de altura (D_P), duração (D_D) e intervalo entre notas (D_{IOI}) são 0.0001±0.016, 0.27±0.01 e 0.83±0.02, respectivamente.

Dentro das oito métricas apresentadas, o projeto apresentou valores promissores em três delas quando comparado aos demais trabalhos. Quando comparado ao trabalho 1, apresentou uma diferença ao conjunto de dados de teste inferior na métrica *IOI*, apresentando

uma maior semelhança ao seu respectivo conjunto de dados de teste, mesmo que ligeiramente em sua maioria. Adicionalmente, demonstra uma distância de Wasserstein inferior considerando o *NLH*.

Já em comparação com o trabalho 2, apresenta métricas favoráveis apenas em uma das três comparações. Na distribuição de intervalo entre notas (D_{IOI}), possui um valor sutilmente superior ao apresentado no outro projeto, indicando uma leve vantagem no contexto desta *feature*.

O projeto obteve resultados relevantes, mas deve ser notada a potencial influência do uso de um conjunto de dados com sons de bateria para o treinamento e teste nas métricas. Algumas das bibliotecas utilizadas retornavam valores zerados em algumas métricas relacionadas à altura dos arquivos, mencionado anteriormente no capítulo 4.7.2.

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
RL-Duet	**+0.12**	**-0.48**	-0.09	**0.0057**	0.042	-	-	-
PopMAG**	-	-	-	-	-	**0.58±0.01**	**0.55±0.01**	0.72±0.01
Este trabalho	+1.90	+1.13	**+0.015**	0.083	**0.015**	0.0001±0.016	0.27±0.01	**0.83±0.02**

Tabela 9 — Tabela comparativa das métricas entre os artigos coletados e este trabalho

** Métricas baseadas neste conjunto de dados*

*** Métricas da tarefa de piano para outras categorias de acompanhamento*

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Bach Chorale	3.25	4.57	3.84	-	-
Este trabalho	3.15	7.71	0.13	-	-

Tabela 10 — Tabela comparativa entre os conjuntos de dados analisados

5 CONCLUSÃO

Este projeto teve a intenção de considerar outras pesquisas e modelos que aplicaram os conceitos de acompanhamento e geração musical, porém focado em batidas de bateria e gerar melodias de piano para o acompanhamento, especificando o modelo elaborado.

Consequentemente, é possível concluir que o projeto atingiu seu objetivo geral. Os objetivos específicos, similarmente, foram cumpridos. O estado da arte em Machine Learning para geração musical foi reconhecido por meio dos trabalhos relacionados, que foram utilizados para assimilar os métodos e sugerir o modelo de geração do projeto deste trabalho. Além disso, diversas métricas foram aplicadas, analisadas e os resultados finais foram levantados e comparados aos trabalhos relacionados.

No decorrer do desenvolvimento do modelo, um tanto de adversidades foram encontradas. A primeira delas foi a falta de compatibilidade de algumas das ferramentas com a nova versão padrão do Python, 3. Além disso, diversas mudanças na biblioteca original e nas ferramentas de métricas precisam ser realizadas para o pleno funcionamento. A falta de estudos com algumas das ferramentas utilizadas também foi um fator adverso. É um tema bastante embrionário, porém extremamente promissor e brilhante.

A comparação com os trabalhos relacionados foi complexa, visto que muitos trabalhos apresentam suas métricas de maneira diversa, ou métricas completamente diferentes. Isto inclusive é um assunto abordado em diversos dos estudos utilizados para referência e anteriormente neste trabalho. Apesar do estudo de Yang e Lerch (2020) definir um bom padrão, é recente e ainda não foi amplamente difundido e adotado. Nos estudos em que é utilizado, pode ser parcialmente aplicado ou ter seus valores representados de formas variadas.

Deste modo, baseado na execução deste trabalho ficam evidentes algumas oportunidades de futuras evoluções que podem ser elaboradas e que estão diretamente relacionadas ao trabalho atual. Estes pontos estão brevemente descritos abaixo.

Visando obter resultados mais confiáveis e precisos, é recomendado a avaliação quantitativa de outros tipos de modelo. Neste projeto, foi considerado aquilo já desenvolvido pelos responsáveis pela biblioteca Magenta, baseando-se nos seus experimentos e resultados para tomar decisões que poderiam levar em conta uma análise própria e mais profunda. Isto poderia ser alcançado treinando outros modelos e comparando métricas semelhantes às apresentadas neste trabalho.

Apesar da interface gráfica ter sido desenvolvida e conectada com o modelo treinado, ela ainda pode ser aprimorada . Isto pode ter estar relacionado a biblioteca Magenta.js e como ela interpreta o modelo customizado desenvolvido ao longo deste trabalho, mas uma avaliação mais assertiva deve ser realizada para encontrar de fato os problemas e possíveis soluções.

Ademais, o modelo pode ser aprimorado e expandido. Pode ser estudada uma maneira de receber e gerar sons de diversos outros instrumentos, tendo assim uma maior variedade musical e aplicabilidade prática. O modelo desenvolvido neste trabalho foca em receber batidas de bateria e gerar melodias de piano. Isto provavelmente implicaria em ainda mais mudanças na biblioteca Magenta e no processo de treinamento.

Assim como o ponto anterior, uma integração do modelo com ferramentas musicais poderia aumentar drasticamente a aplicabilidade e uso prático. Isto já foi realizado com outros modelos criados pela equipe responsável pela biblioteca Magenta na ferramenta Ableton, o que pode facilitar o aprimoramento do modelo desenvolvido neste trabalho. Outra ferramenta semelhante ao Ableton que poderia ser utilizada para integração é o Fruity Loops, ou, como é mais comumente conhecido, FL Studio.

O conceito de estrutura de longo prazo, mencionado ao longo deste projeto, também pode ser considerado com mais importância em futuras evoluções. Como o modelo foi treinado para gerar apenas dois compassos por arquivo MIDI, a estrutura de longo prazo não pôde ser avaliada com confiabilidade nos resultados. Um bom caminho pode envolver o aumento no número dos compassos considerados ao treinar o modelo geracional.

Também poderia ser de interesse aplicar uma avaliação subjetiva com seres humanos, com conhecimento musical e/ou não, para validar os resultados do estudo além das métricas

objetivas e olhar pessoal potencialmente enviesado. Esse tipo de avaliação é comumente utilizada em outros estudos criativos voltados ao campo musical.

REFERÊNCIAS BIBLIOGRÁFICAS

DONG, Hao-Wen et al. MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. **Thirty-Second AAAI Conference on Artificial Intelligence**. 2018.

EGHBAL-ZADEH, Hamid et al. A GAN based Drum Pattern Generation UI Prototype. **Artificial Intelligence**, v. 94, p. 1459, 1994.

HUANG, Yu-Siang; CHOU, Szu-Yu; YANG, Yi-Hsuan. Pop music highlighter: Marking the emotion keypoints. **arXiv preprint arXiv:1802.10495**, 2018.

LU, Chien-Yu et al. Play as you like: Timbre-enhanced multi-modal music style transfer. **Proceedings of the AAAI Conference on Artificial Intelligence**. 2019. p. 1061-1068.

ROBERTS, Adam et al. A hierarchical latent vector model for learning long-term structure in music. In: **International conference on machine learning**. PMLR, 2018. p. 4364-4373.

GILLICK, Jon et al. Learning to groove with inverse sequence transformations. In: **International Conference on Machine Learning**. PMLR, 2019. p. 2269-2279.

SENN, Olivier et al. Groove in drum patterns as a function of both rhythmic properties and listeners' attitudes. **PloS one**, v. 13, n. 6, p. e0199604, 2018.

GOYA-MARTINEZ, Mariana. The emulation of emotions in artificial intelligence: Another step into anthropomorphism. **Emotions, Technology, and Design**. Academic Press, 2016. p. 171-186.

FONTELLES, Mauro José et al. Metodologia da pesquisa científica: diretrizes para a elaboração de um protocolo de pesquisa. **Revista Paraense de Medicina**, v. 23, n. 3, p. 1-8, 2009.

JIANG, Nan et al. RL-Duet: Online Music Accompaniment Generation Using Deep Reinforcement Learning. **Proceedings of the AAAI Conference on Artificial Intelligence**. 2020. p. 710-718.

JAQUES, Natasha et al. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. **International Conference on Machine Learning**. PMLR, 2017. p. 1645-1654.

CUTHBERT, Michael Scott; ARIZA, Christopher. **music21: A toolkit for computer-aided musicology and symbolic music dat**a. 2010.

HADJERES, Gaëtan; PACHET, François; NIELSEN, Frank. Deepbach: a steerable model for bach chorales generation. **International Conference on Machine Learning**. PMLR, 2017. p. 1362-1371.

BRIOT, Jean-Pierre; HADJERES, Gaëtan; PACHET, François-David. Deep learning techniques for music generation--a survey. **arXiv preprint arXiv:1709.01620**, 2017.

YANG, Li-Chia; LERCH, Alexander. On the evaluation of generative models in music. **Neural Computing and Applications**, v. 32, n. 9, p. 4773-4784, 2020.

REN, Yi et al. Popmag: Pop music accompaniment generation. **Proceedings of the 28th ACM International Conference on Multimedia**. 2020. p. 1198-1206.

DAI, Zihang et al. Transformer-xl: Attentive language models beyond a fixed-length context. **arXiv preprint arXiv:1901.02860**, 2019.

RAFFEL, Colin. **Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching**. 2016. Tese de Doutorado. Columbia University.

HUANG, Cheng-Zhi Anna et al. Music transformer: Generating music with long-term structure. **International Conference on Learning Representations**. 2018.

HUANG, Yu-Siang; YANG, Yi-Hsuan. Pop music transformer: Generating music with rhythm and harmony. **arXiv preprint arXiv:2002.00212**, 2020.

ZHU, Hongyuan et al. Xiaoice band: A melody and arrangement generation framework for pop music. **Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. 2018. p. 2837-2846.

VASWANI, Ashish et al. Attention is all you need. **Advances in neural information processing systems**. 2017. p. 5998-6008.

BROWN, Tom B. et al. Language models are few-shot learners. **arXiv preprint arXiv:2005.14165**, 2020.

SONG, Kaitao et al. Mass: Masked sequence to sequence pre-training for language generation. **arXiv preprint arXiv:1905.02450**, 2019.

MICHIE, Donald et al. Machine learning. **Neural and Statistical Classification**, v. 13, n. 1994, p. 1-298, 1994.

FAUL, Anita C. **A Concise Introduction to Machine Learning**. CRC Press, 2019.

KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237-285, 1996.

SZEPESVÁRI, Csaba. Algorithms for reinforcement learning. **Synthesis lectures on artificial intelligence and machine learning**, v. 4, n. 1, p. 1-103, 2010.

XU, Siwei. A Structural Overview of Reinforcement Learning Algorithms. Junho 2020. **Towards Data Science**. <https://towardsdatascience.com/an-overview-of-classic-reinforcement-learning-algorithms-part-1-f79c8b87e5af>. Acessado em 18/01/2021.

KONDA, Vijay R.; TSITSIKLIS, John N. Actor-critic algorithms. **Advances in neural information processing systems**. 2000. p. 1008-1014.

BAHDANAU, Dzmitry et al. An actor-critic algorithm for sequence prediction. **arXiv preprint arXiv:1607.07086**, 2016.

SCHULMAN, John et al. High-dimensional continuous control using generalized advantage estimation. **arXiv preprint arXiv:1506.02438**, 2015.

GHAHRAMANI, Zoubin. Unsupervised learning. **Summer School on Machine Learning**. Springer, Berlin, Heidelberg, 2003. p. 72-112.

CELEBI, M. Emre; AYDIN, Kemal (Ed.). **Unsupervised learning algorithms**. Berlin: Springer International Publishing, 2016.

DAYAN, Peter; SAHANI, Maneesh; DEBACK, Grégoire. Unsupervised learning. **The MIT encyclopedia of the cognitive sciences**, p. 857-859, 1999.

KINGMA, Diederik P.; WELLING, Max. Auto-encoding variational bayes. **arXiv preprint arXiv:1312.6114**, 2013.

KINGMA, Diederik P.; WELLING, Max. An introduction to variational autoencoders. **arXiv preprint arXiv:1906.02691**, 2019.

BOWMAN, Samuel R. et al. Generating sentences from a continuous space. **arXiv preprint arXiv:1511.06349**, 2015.

VAHDAT, Arash; KAUTZ, Jan. Nvae: A deep hierarchical variational autoencoder. **arXiv preprint arXiv:2007.03898**, 2020.

LI, Xiaopeng; SHE, James. Collaborative variational autoencoder for recommender systems. **Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining**. 2017. p. 305-314.

WEI, I.-Chieh; WU, Chih-Wei; SU, Li. Generating Structured Drum Pattern Using Variational Autoencoder and Self-similarity Matrix. **ISMIR**. 2019. p. 847-854.

YAMSHCHIKOV, Ivan P.; TIKHONOV, Alexey. Music generation with variational recurrent autoencoder supported by history. **SN Applied Sciences**, v. 2, n. 12, p. 1-7, 2020.

HADJERES, Gaëtan. Interactive deep generative models for symbolic music. 2018. Tese de Doutorado.

FU, Yongjian. Data mining. **IEEE Potentials**, v. 16, n. 4, p. 18-20, 1997.

LI, Tao; OGIHARA, Mitsunori; TZANETAKIS, George (Ed.). Music data mining. **CRC Press**, 2011.

NIERHAUS, Gerhard. Algorithmic composition: paradigms of automated music generation. **Springer Science & Business Media**, 2009.

AGGARWAL, Charu C. Data mining: the textbook. **Springer**, 2015.

LI, Tao; OGIHARA, Mitsunori; TZANETAKIS, George. Guest editorial: special section on music data mining. **IEEE Transactions on Multimedia**, v. 16, n. 5, p. 1185-1187, 2014.

WÜLFING, Jan; RIEDMILLER, Martin A. Unsupervised Learning of Local Features for Music Classification. **ISMIR**. 2012. p. 139-144.

ROCHE, Fanny et al. Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models. **arXiv preprint arXiv:1806.04096**, 2018.

BRUNNER, Gino et al. MIDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer. **arXiv preprint arXiv:1809.07600**, 2018.

OJA, Erkki; HYVARINEN, A. Independent component analysis: algorithms and applications. **Neural networks**, v. 13, n. 4-5, p. 411-430, 2000.

XIAO, Linxing et al. Using Statistic Model to Capture the Association between Timbre and Perceived Tempo. **ISMIR**. 2008. p. 659-662.

KNOPKE, Ian. Sound, Music and Textual Associations on the World Wide Web. **ISMIR**. 2004.

KUO, Fang-Fei et al. Emotion-based music recommendation by association discovery from film music. **Proceedings of the 13th annual ACM international conference on Multimedia**. 2005. p. 507-510.

FOOTE, Jonathan. A similarity measure for automatic audio classification. **Proc. AAAI 1997 Spring Symposium on Intelligent Integration and Use of Text, Image, Video, and Audio Corpora**. 1997.

TZANETAKIS, George; COOK, Perry. Musical genre classification of audio signals. **IEEE Transactions on speech and audio processing**, v. 10, n. 5, p. 293-302, 2002.

LIU, Dan; LU, Lie; ZHANG, Hong-Jiang. Automatic mood detection from acoustic music data. 2003.

TSAI, Wei-Ho; RODGERS, Dwight; WANG, Hsin-Min. Blind clustering of popular music recordings based on singer voice characteristics. **Computer Music Journal**, v. 28, n. 3, p. 68-78, 2004.

SHAO, Xi et al. Automatic music summarization in compressed domain. **2004 IEEE International Conference on Acoustics, Speech, and Signal Processing**. IEEE, 2004. p. iv-iv.

PILHOFER, Michael; DAY, Holly. **Music theory for dummies**. John Wiley & Sons, 2019.

LASKE, Otto. Composition theory: An enrichment of music theory. **Journal of New Music Research**, v. 18, n. 1-2, p. 45-59, 1989.

CLENDINNING, Jane Piper; MARVIN, Elizabeth West. **The musician's guide to theory and analysis**. WW Norton & Company, 2016.

MED, Bohumil. Teoria da música. Brasília: Musimed, 1996.

SIMPLIFYING THEORY. **Timbre Definition**. 2014. Disponível em: https://www.simplifyingtheory.com/timbre/. Acesso em: 24 jan. 2021.

DESCOMPLICANDO A MÚSICA. **Intensidade do som: o que é intensidade do som?**. Disponível em: https://www.descomplicandoamusica.com/intensidade-do-som/. Acesso em: 24 jan. 2021.

BENWARD, Bruce; SAKER, Marilyn. **Music**: in theory and practice. 7. ed. Nova York: McGraw-Hill, 2003.

DELONE, Richard et al. **Aspects of twentieth-century music**. Nova Jersey: Prentice Hall, p. 208-269, 1975.

MARTINEAU, Jason. **The Elements of Music**: Melody, Rhythm, and Harmony. Bloomsbury Publishing USA, 2008.

TERHARDT, Ernst. The concept of musical consonance: A link between music and psychoacoustics. **Music perception**, v. 1, n. 3, p. 276-295, 1984.

RASHAD, Fathy. **Generative Modeling with Variational Auto Encoder (VAE)**. 2020. Disponível em: https://medium.com/vitrox-publication/generative-modeling-with-variational-auto-encoder-vae-fc449be9890e. Acesso em: 11 fev. 2021.

HOFFBERG, Mark B. **Bibliographic music data base with normalized musical themes**. U.S. Patent n. 5,963,957, 5 out. 1999.

HADJERES, Gaëtan; PACHET, François; NIELSEN, Frank. Deepbach: a steerable model for bach chorales generation. **International Conference on Machine Learning**. PMLR, 2017. p. 1362-1371.

DOWNIE, J. Stephen. Music information retrieval. **Annual review of information science and technology**, v. 37, n. 1, p. 295-340, 2003.

JUSLIN, Patrik N.; SLOBODA, John (Ed.). **Handbook of music and emotion: Theory, research, applications**. Oxford University Press, 2011.

WERBOCK, Jeffrey. **Are the emotions that accompany different musical styles and effects nature or nurture? In other words, could two people with dramatically different life experiences feel completely different things when listening to the same piece of music?**. 2019. Disponível em: https://www.quora.com/Are-the-emotions-that-accompany-different-musical-styles-and-effects-nature-or-nurture-In-other-words-could-two-people-with-dramatically-different-life-experiences-feel-completely-different-things-when-listening-to. Acesso em: 31 ago. 2021.

MILLER, Michael. **The complete idiot's guide to music theory**. Penguin, 2005.

KUHLMAN, Dave. **A python book: Beginning python, advanced python, and python exercises**. Lutz: Dave Kuhlman, 2009.

GRUS, Joel. **Data science from scratch: first principles with python**. O'Reilly Media, 2019.

ROBERTS, Adam et al. **Magenta studio: Augmenting creativity with deep learning in ableton live**. 2019.

VAN KRANENBURG, Peter; JANSSEN, Berit; VOLK, Anja. **The Meertens Tune Collections: The Annotated Corpus (MTC-ANN) versions 1.1 and 2.0. 1**. Meertens Online Reports, v. 2016, n. 1, 2016.

WAITE, Elliot. **Generating Long-Term Structure in Songs and Stories**. 2016. Disponível em: https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn. Acesso em: 19 set. 2021.

APPLE COMPUTER INC (Estados Unidos). **Speech Attributes**. 1996. Disponível em: http://mirror.informatimago.com/next/developer.apple.com/documentation/mac/Sound/Sound-190.html. Acesso em: 30 set. 2021.

KINGMA, Diederik P.; BA, Jimmy. **Adam: A method for stochastic optimization**. arXiv preprint arXiv:1412.6980, 2014.

ROBERTS, Adam; HAWTHORNE, Curtis; SIMON, Ian. **Magenta.js: a JavaScript API for augmenting creativity with deep learning**. 2018.

RUÉ VILÀ, Aleix. **Drums generation from a tapped pattern**. 2020. Trabalho de Conclusão de Curso. Universitat Politècnica de Catalunya.

DUBREUIL, Alexandre. **Hands-On Music Generation with Magenta**. Packt Publishing, 2020. Disponível em: https://github.com/PacktPublishing/hands-on-music-generation-with-magenta. Acesso em: 16 jan. 2022.

CROCKFORD, Douglas. **The application/json media type for javascript object notation (json)**. 2006.

CALLENDER, Lee; HAWTHORNE, Curtis; ENGEL, Jesse. **Improving perceptual quality of drum transcription with the expanded groove midi dataset**. arXiv preprint arXiv:2004.00188, 2020.

CHETLUR, Sharan et al. **cudnn: Efficient primitives for deep learning**. arXiv preprint arXiv:1410.0759, 2014.

KASHYAP, Ravi. **The universal language: mathematics or music?**. Journal for Multicultural Education, 2021.

ONGSULEE, Pariwat. **Artificial intelligence, machine learning and deep learning**. In: 2017 15th international conference on ICT and knowledge engineering (ICT&KE). IEEE, 2017. p. 1-6.

BROWNLEE, Jason. **How Much Training Data is Required for Machine Learning?**. 2017. Disponível em: machinelearningmastery.com/much-training-data-required-machine-learning/. Acesso em: 1 ago. 2022.

PU, Yunchen et al. **Variational autoencoder for deep learning of images, labels and captions**. Advances in neural information processing systems, v. 29, 2016.

VINYALS, Oriol et al. **Show and tell: Lessons learned from the 2015 mscoco image captioning challenge**. IEEE transactions on pattern analysis and machine intelligence, v. 39, n. 4, p. 652-663, 2016.

MANSIMOV, Elman et al. **Generating images from captions with attention**. arXiv preprint arXiv:1511.02793, 2015.

REED, Scott et al. **Generative adversarial text to image synthesis**. In: International conference on machine learning. PMLR, 2016. p. 1060-1069.

REED, Scott E. et al. **Learning what and where to draw**. Advances in neural information processing systems, v. 29, 2016.

GIBIANSKY, Andrew et al. **Deep voice 2: Multi-speaker neural text-to-speech**. Advances in neural information processing systems, v. 30, 2017.

OORD, Aaron van den et al. **Wavenet: A generative model for raw audio**. arXiv preprint arXiv:1609.03499, 2016.

APÊNDICES

APÊNDICE A - RESULTADO DA EXECUÇÃO DA MGEVAL

Conforme visto no capítulo 4.7.2, a execução das funcionalidades da caixa de ferramentas MGEval resulta em um extenso JSON com todas as métricas geradas durante o processo. Respectivamente por posição, para cada métrica, estão dispostas: a média da métrica calculada para o conjunto de dados de teste; o desvio padrão da mesma métrica; a média da métrica calculada para o conjunto de dados gerados; o desvio padrão da mesma métrica; a divergência de Kullback-Leibler entre a distância *intra-set* do conjunto de dados de teste e a distância *inter-set*; a área sobreposta entre a distância *intra-set* do conjunto de dados de teste e a distância *inter-set;* a divergência de Kullback-Leibler entre a distância *intra-set* do conjunto de dados gerado e a distância *inter-set*; a área sobreposta entre a distância *intra-set* do conjunto de dados gerado e a distância *inter-set*; e a distância de Wasserstein, calculada apenas para *PCH* e *NLH*.

```
{
  "total_used_pitch": [
    [0.0],
    [0.0],
    [10.1],
    [1.374772708486752],
    0,
    0.9499584497297907,
    0.05298666688356432,
    0.00018469331175657521,
    0
  ],
  "pitch_range": [0, 0, [31.8], [2.85657137141714], 0, 0, 0, 0, 0],
  "avg_pitch_shift": [
    [7.712163799088228],
    [3.6389212055693276],
    [9.567081248861808],
    [1.9466274877434597],
    0.1081756744737632,
    0.7006210654780691,
    0.09839588457623547,
    0.7112623339037759,
    0
  ],
  "avg_IOI": [
    [0.12781841504082442],
    [0.0296016668612053],
    [0.10788158882359859],
    [0.014296003680925071],
    0.0011894507102116515,
    0.8682731525669002,
    0.03811086048437187,
    0.6461807415044284,
    0
  ],
  "total_used_note": [
    [451.5],
    [467.71299960552733],
    [36.3],
    [5.197114584074513],
    0.05089514632159842,
    0.6685007617326519,
    0.12451357520752462,
    0.022940715729437965,
    0
  ],
```

```
  "bar_used_pitch": [
    [[2.5], [3.8]],
    [[1.5], [2.1354156504062622]],
    [[6.3], [7.4]],
    [[1.7349351572897473], [1.428285685708 57]],
    0.027855911152587405,
    0.6535586018125775,
    0.13396682444037855,
    0.5226755364954011,
    0
  ],
  "bar_used_note": [
    [[7.0], [14.7]],
    [[4.381780460041329], [7.416872656315463]],
    [[10.0], [18.6]],
    [[2.6076809620810595], [4.6303347611160905]],
    0.014087883123786497,
    0.8618602549530294,
    0.0818183493293327,
    0.7491088440331101,
    0
  ],
  "total_pitch_class_histogram": [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [
      0.34187677371020875, 0.0040208936071729874, 0.21256990154334252,
      0.09784343380362828, 0.046503204937523315, 0.002450066133351158,
      0.03208548536671153, 0.00460415090120792, 0.1700224624683005,
      0.0012155830554822736, 0.08660129051028648, 0.00020675396278428668
    ],
    [
      0.0743107744155862, 0.0050799771019921 89, 0.08649147604630127,
      0.038470901976538296, 0.027814812872038013, 0.0040708531388204535,
      0.015293574262575561, 0.006628367115848823, 0.056892486753413234,
      0.001735845887161749, 0.03352201146553294, 0.0006202618883528601
    ],
    0,
    0.12309694422802882,
    0.03183977962741551,
    0.0017839566009244618,
    0.08333333333333334
  ],
```

```
"bar_pitch_class_histogram": [
  [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  ],
  [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  ],
  [
    [
      0.39743013399418714, 0.004105487997157001, 0.20014658728804752,
      0.08594324184474215, 0.04596588444845528, 0.0027562792874510976,
      0.021769505032957832, 0.0017972310124180421, 0.15113957080596208,
      0.0016715717269109357, 0.08727450656171085, 0.0
    ],
    [
      0.2748031749784067, 0.003785621556831588, 0.2298370105739774,
      0.10865311277504841, 0.0494749629408049, 0.0022838271810342392,
      0.04529843515926039, 0.007775963560408749, 0.18805620306880969,
      0.0010324299571208525, 0.08848970410816975, 0.0005095541401273885
    ]
  ],
  [
    [
      0.09401684703000164, 0.006046736024058873, 0.11942295938378954,
      0.048839151451778515, 0.047396260418360744, 0.007087826201922716,
      0.02049422998236176, 0.0036446427442808505, 0.0935901280676174,
      0.003708356382703637 3, 0.05944338162425899, 0.0
    ],
    [
      0.15187149029315974, 0.004641995904048458, 0.13752296315581078,
      0.06915879901341522, 0.06126501107856589, 0.005372246816860451,
      0.04259499934668759, 0.011976911415780962, 0.06733898387241491,
      0.0020650747688194925, 0.07456385497336943, 0.0015286624203821656
    ]
  ],
  0,
  0.23171352814770887,
  0.012274154544407742,
  0.15900209448099104,
  0
],
```

```
"note_length_hist": [
  [
    0.0, 0.0, 0.0, 0.0, 0.9120819848975188, 0.0, 0.0, 0.0,
    0.080906148867313 93, 0.0, 0.0, 0.007011866235167206
  ],
  [
    0.0, 0.0, 0.0, 0.0, 0.2637540453074434, 0.0, 0.0, 0.0,
    0.24271844660194172, 0.0, 0.0, 0.02103559870550162
  ],
  [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
  0.1616519507656133,
  0.4009448251468232,
  0,
  0.3236006889206613,
  0.014653002517080191
],
```

```
  "pitch_class_transition_matrix": [
    [
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    ],
    [
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    ],
    [
      [
        0.07710131856636247, 0.004759693522173206, 0.05165801153735611,
        0.019855374518628535, 0.01411522343894162, 0.0058823529411764705,
        0.016794196221892845, 0.004651162790697674, 0.03982001781452994,
        0.005263157894736842, 0.024383496162904774, 0.005263157894736842
      ],
      [
        0.006346078044934962, 0.0, 0.00386986301369863, 0.011272712604461225,
        0.0, 0.0, 0.0, 0.0, 0.004020496649586125, 0.0, 0.005266757865937072, 0.0
      ],
      [
        0.0565041218951441, 0.004311039484286865, 0.07789627302660443,
        0.025678594440535173, 0.005143447919359007, 0.0, 0.018145888492383255,
        0.005907289102416314, 0.0196043643007658, 0.009036742800397218,
        0.01279074412597725, 0.005263157894736842
      ],
      [
        0.014294196221892846, 0.0, 0.020583752088748093, 0.005120678408349641,
        0.004267744833782569, 0.0, 0.001694915254237288, 0.0,
        0.009141175057935765, 0.0, 0.011641175057935766, 0.0
      ],
```

```
      [
        0.010273699572306724, 0.0, 0.014160315288906261, 0.0, 0.0, 0.0, 0.0025,
        0.0, 0.0, 0.0, 0.0, 0.0
      ],
      [
        0.0, 0.0, 0.001694915254237288, 0.001694915254237288, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0
      ],
      [
        0.01710536935559237, 0.0025, 0.030941222732569218, 0.01082793616267276,
        0.005322128851540616, 0.0, 0.016101459446641 45, 0.0018867924528301887,
        0.015972046438361982, 0.0, 0.008834586466165414, 0.0
      ],
      [
        0.0018867924528301887, 0.0, 0.006537955243527863, 0.0,
        0.0018867924528301887, 0.0, 0.0, 0.0018867924528301887, 0.0,
        0.0018867924528301887, 0.0, 0.0
      ],
      [
        0.033413953355036906, 0.001694915254237288, 0.029658064582310767,
        0.02211415842320299, 0.004651162790697674, 0.0, 0.004759693522173206,
        0.004651162790697674, 0.008424747696358051, 0.0, 0.015212603629364338,
        0.0
      ],
      [
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001694915254237288, 0.0, 0.0, 0.0, 0.0,
        0.0
      ],
      [
        0.020711952653896918, 0.005390359663284755, 0.02647608568772384,
        0.019836249568320478, 0.0046360917248255236, 0.0, 0.010502883922134101,
        0.0037735849056603774, 0.008331536133872991, 0.007577268195413758,
        0.00523972602739726, 0.0
      ],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    ],
    [
      [
        0.03095545860678102, 0.010536606336948695, 0.0417140974744634,
        0.019265746669793103, 0.0206746612151485, 0.01764705882352941,
        0.012124661123715904, 0.013953488372093023, 0.029394128970150077,
        0.015789473684210527, 0.029893681544119765, 0.015789473684210527
      ],
      [
        0.014310409627042922, 0.0, 0.00814183069206197, 0.0179031456453965, 0.0,
        0.0, 0.0, 0.0, 0.00816371680945975, 0.0, 0.010623075833153741, 0.0
      ],
      [
        0.03500340093096095, 0.0093104981413098, 0.06899464055451855,
        0.02243897775620899, 0.011606447374891027, 0.0, 0.025665188547055882,
        0.009138624541769511, 0.01882747669474229, 0.018377840466470882,
        0.014509627213381664, 0.015789473684210527
```

```
      ],
      [
        0.012736890848021428, 0.0, 0.020413396434998773, 0.010272730021463117,
        0.008606715635939177, 0.0, 0.005084745762711864, 0.0,
        0.01144551936218579, 0.0, 0.011897228172162246, 0.0
      ],
      [
        0.01098808952307263, 0.0, 0.019591576844148105, 0.0, 0.0, 0.0,
        0.007500000000000001, 0.0, 0.0, 0.0, 0.0, 0.0
      ],
      [
        0.0, 0.0, 0.005084745762711864, 0.005084745762711864, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0
      ],
      [
        0.018602809492969344, 0.007500000000000001, 0.0333307178610368,
        0.016424272667605357, 0.010717717910201112, 0.0, 0.024511302099175444,
        0.005660377358490565, 0.030937177310238304, 0.0, 0.018069571456714673,
        0.0
      ],
      [
        0.005660377358490565, 0.0, 0.014463338200080295, 0.0,
        0.005660377358490565, 0.0, 0.0, 0.005660377358490565, 0.0,
        0.005660377358490565, 0.0, 0.0
      ],
      [
        0.029909069385040007, 0.005084745762711864, 0.02736895800636244,
        0.03442910385433446, 0.013953488372093023, 0.0, 0.010536606336948695,
        0.013953488372093023, 0.016963378487515213, 0.0, 0.013160118013680996,
        0.0
      ],
      [
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.005084745762711864, 0.0, 0.0, 0.0, 0.0,
        0.0
      ],
      [
        0.020086293413342535, 0.008515863721186276, 0.02451480382571557,
        0.02511012710075453, 0.009681901725430058, 0.0, 0.017443013690858128,
        0.01132075471698113, 0.010893420407694329, 0.017813847005375014,
        0.010493152920347415, 0.0
      ],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    ],
    0,
    0.20713221646201674,
    0.09640531851501766,
    0.8081262771809354,
    0
  ],
```

```
"note_length_transition_matrix": [
  [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 359.1, 0.0, 0.0, 0.0, 9.4, 0.0, 0.0, 0.6],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 7.0, 0.0, 0.0, 0.0, 62.1, 0.0, 0.0, 5.8],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 3.5, 0.0, 0.0, 0.0]
  ],
  [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [
      0.0, 0.0, 0.0, 0.0, 454.9941647977477, 0.0, 0.0, 0.0,
      28.199999999999996, 0.0, 0.0, 1.8
    ],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [
      0.0, 0.0, 0.0, 0.0, 21.0, 0.0, 0.0, 0.0, 186.29999999999995, 0.0, 0.0,
      17.4
    ],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 9.0, 0.0, 0.0, 0.0, 10.5, 0.0, 0.0, 0.0]
  ],
  [
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 35.3, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  ],
```

```
    [
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [
        0.0, 0.0, 0.0, 0.0, 5.197114584074513, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
      ],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    ],
    0.0603599951974821,
    0.6737593646491935,
    0.10578676342202498,
    0.02364416450771666,
    0
  ]
}
```


Figuras 31, 32, 33, 34, 35, 36, 37, 38 e 39 — JSON retornado pela execução do arquivo __main__.py

APÊNDICE B - CÓDIGOS E FERRAMENTAS UTILIZADAS NO PROJETO

- O código da biblioteca Magenta modificada está disponível em: https://github.com/mbrosowicz/magenta
- O código utilizado da Magenta JS pode ser encontrado em: https://github.com/mbrosowicz/magenta-js
- O código front-end do protótipo desenvolvido está disponível em: https://github.com/mbrosowicz/melodify
- O código back-end do protótipo desenvolvido pode ser encontrado em: https://github.com/mbrosowicz/melodify-backend
- O modelo treinado para uso em conjunto com a biblioteca Magenta está disponível para download em: https://drive.google.com/drive/folders/1-q3fhhcfTaK-kycPKSJpFxpwrWOH_kvh?usp=sharing
- Os arquivos MIDI utilizados para construir resultados e fazer comparações podem ser encontrados em: https://drive.google.com/drive/folders/1Z-RO5rIEDymXzUs-KsjTWcWeK_4brCkk?usp=sharing
- O conjunto de dados utilizado está disponível em: https://magenta.tensorflow.org/datasets/groove

Geração de Música com Machine Learning

Marcelo Brosowicz de Paulo

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88010-970 – Florianópolis – SC – Brazil

brosowiczm@gmail.com

Abstract. *Music composition and production are processes mostly performed by humans and facilitated by computers, but dependent on the human understanding of emotional and creative areas involved in the final product. The main goal of this work is to research and analyze the musical production from user input (such as melodies, songs) using machine learning techniques, in order to propose a prototype application capable of generating melodies with notes that accompany a drum beat supplied to the program by the user in MIDI format. Experiments are conducted for generating piano melodies and the results of said experiments are also duly analyzed and compared with similar experiments.*

Resumo. *A composição e produção musical são processos majoritariamente realizados por humanos e facilitados por computadores, mas dependentes da compreensão humana de áreas emocionais e criativas envolvidas no produto final. O principal objetivo deste trabalho consiste na pesquisa e análise da produção musical a partir de entradas do usuário (como melodias, músicas) utilizando técnicas de aprendizado de máquina, com intuito de propor um protótipo de aplicação capaz de gerar melodias com notas que acompanhem uma batida de bateria fornecida ao programa pelo usuário em formato MIDI. Experimentos são realizados para geração de melodias de piano e os resultados de ditos experimentos são também devidamente analisados e comparados com experimentos semelhantes.*

1. Introdução

A música é uma linguagem universal, conforme concluído por Kashyap (2021). Seres humanos a utilizam desde a era paleolítica para se comunicarem e expressarem sentimentos entre si. Ao longo da história, compositores surpreendentes e cativantes surgiram sem aparentes padrões e razões claras para seus sucessos, trazendo consigo muitas vezes estilos, conceitos e técnicas nunca vistas e praticamente impossíveis de serem previstas. Músicos estes que revolucionaram o processo criativo com novos estilos musicais compondo peças inovadoras e/ou muito precisos na criação de peças com uma abundância de estrutura musical subjacente. É possível então que o computador também consiga aprender a criar essa estrutura musical?

O aprendizado de máquina (ML) está relacionado a um conceito muito anterior chamado inteligência artificial (IA) — consolidado por volta da década de 50 — com redes neurais simples e modelos ainda muito novos, que vieram evoluindo exponencialmente com a história e popularizando-se cada vez mais como um importante meio de fornecer aos sistemas a autonomia, ou seja, a capacidade de aprender e

melhorar automaticamente a partir da experiência sem ser explicitamente programado, como visto na abordagem do estudo por Ongsulee (2017). O aprendizado de máquina centra-se no desenvolvimento de “softwares” que podem acessar dados e usá-los para aprender por si mesmos.

Recentemente, houve grande progresso na aplicação do aprendizado de máquina a uma gama de problemas relacionados à música, como "thumb-nailing" (HUANG, CHOU e YANG, 2018), geração de música (ROBERTS et al., 2018) e transferência de estilo (LU, 2019). Para demonstrar o resultado desses modelos de aprendizagem de máquina no contexto criativo e musical, os pesquisadores geralmente colocam a saída de áudio como o resultado nos sites dos projetos.

Apesar do processo realizado e criatividade ser um dos assuntos mais populares em inteligência artificial, especialistas se perguntam, no entanto, até onde a IA pode ou deve ir no processo criativo. A IA já ajudou a escrever baladas pop, imitou os estilos de grandes pintores e informou decisões criativas na produção de filmes, mas quão presente no contexto criativo ela deve ou pode estar? Um dos problemas na aplicação do aprendizado de máquina e IA no processo de criação musical é o quão “humanizadas” e bem colocadas serão as saídas, que será abordado neste trabalho.

2. Trabalhos Relacionados

Visando encontrar artigos relacionados ao objetivo geral deste trabalho, foram colhidas pesquisas que, em algum nível do seu escopo, tinham preocupação em gerar acompanhamento musical para determinada entrada do usuário. Essa seção apresenta os dois artigos que melhor representam a ligação com este trabalho.

2.1. RL-Duet: Online Music Accompaniment Generation Using Deep Reinforcement Learning

Jiang et al. (2020) tiveram como objetivo desenvolver um modelo capaz de criar música colaborativamente com um músico humano, escutando a música tocada e gerando acompanhamento. Para isso, o modelo foi treinado com o conjunto de dados Bach Chorale do Music21, um kit de ferramentas para Musicologia Assistida por Computador criado pelo MIT (CUTHBERT e ARIZA, 2010). Do conjunto de dados, foram utilizados corais com quatro partes monofônicas, no formato SATB (Soprano, Alto, Tenor e Bass) como conjuntos de dados de treinamento e validação, com 327 e 37 corais respectivamente, segundo Jiang et al. (2020).

Para representar a batida e as informações rítmicas adicionais da música, Jiang et al. (2020) utilizam uma lista de subdivisões para codificar a batida em formação, similarmente como acontece no estudo de Hadjeres, Pachet e Nielsen (2017). O tempo é quantificado com dezesseis notas, cada batida sendo subdividida em quatro partes.

O estudo ainda tem modelos de recompensa, considerados pelos autores como a chave para o sucesso do algoritmo. Estes modelos consideram tanto a consistência temporal horizontal (BRIOT, HADJERES e PACHET, 2017) sobre a parte gerada pela máquina, quanto as relações de harmonia vertical (BRIOT, HADJERES e PACHET, 2017) entre as partes humana e máquina (JIANG et al., 2020). A consistência temporal

horizontal está relacionada diretamente à duração das peças, ou seja, a consistência ao longo da sequência temporal. Já a harmonia vertical é relacionada à disposição e combinação de notas em uma determinada etapa temporal. Dessa forma, podemos considerar o eixo horizontal como o eixo temporal e o vertical como harmônico, como apresentado por Briot, Hadjeres e Pachet (2017).

Para validar o projeto, Jiang et al. (2020) comparam o desempenho do RL-Duet com um modelo de base maximum likelihood estimation (MLE) e um modelo de aprendizado por reforço com base em regras. Este modelo com base regras nada mais é que uma implementação própria do SequenceTutor, de Jaques et al. (2017), que usa uma recompensa baseada em modelos, com muitas recompensas baseadas em regras, e os pesos entre elas são ajustados com precisão (JIANG et al., 2020).

Foi observado por Jiang et al. (2020) que existe uma ligeira degeneração da parte gerada por todos os três algoritmos em comparação, considerando que o início da parte gerada sempre foi inicializado com o valor de referência. Ou seja, todos os algoritmos apresentaram algum nível de desvio nos resultados, considerando os valores de entrada. De todo modo, o RL-Duet se mostrou o mais robusto e o que alcança a métrica de avaliação mais próxima do valor de referência. Interessantemente, também foi descoberto que o MLE captura a tendência dessas métricas do conjunto de dados. Entretanto, devido ao viés de exposição (bias), os problemas de geração tendem a se acumular temporalmente, levando a desvios significativos e divergências das métricas objetivas.

Para avaliação subjetiva, foi aplicado um pequeno questionário e comparações em pares com usuários voluntários. É comparado o desempenho do RL-Duet com o modelo de base MLE, visto que o modelo RL-Rules está excluído devido à geração de resultados nada satisfatórios. O questionário possui as seguintes perguntas: 1) Você aprendeu a tocar um instrumento musical (não menos que 5 horas por semana) por mais de 5 anos no total no passado? e 2) Quanto tempo você gasta para ouvir música clássica a cada semana?

Nas comparações, cada usuário recebeu um par de trechos curtos de dueto com um humano. Então, os trechos são randomicamente truncados se baseando em duetos gerados. Após ouvir cada par de amostras de dueto, o participante foi solicitado a escolher qual era o preferido. Cada par de amostras de dueto foi avaliado por 20 sujeitos. 125 sujeitos realizaram o teste e 2000 votos válidos foram coletados. 19 deles (com 233 votos) aprenderam um instrumento musical antes, e 28 deles (com 617 votos) passaram mais de 1 hora ouvindo música clássica por semana (JIANG et al., 2020).

No total, cerca de 58,42% dos sujeitos preferem os duetos gerados pelo RL-Duet aos gerados pelo modelo MLE (JIANG et al., 2020). Se os duetos em comparação se baseiam na mesma parte humana, pode ser difícil diferenciar os dois duetos se o sujeito tem um histórico musical limitado. Já a preferência pelo RL-Duet em relação ao MLE é mais nítida para sujeitos com mais desempenho e experiências auditivas.

2.2. Popmag: Pop music accompaniment generation

Ren et al. (2020) propuseram um modelo Transformer modificado com uma nova representação MIDI, que permite gerar múltiplas faixas simultaneamente em uma única sequência e modelar explicitamente a dependência das notas de diferentes faixas, visando melhorar a harmonia entre a entrada e o acompanhamento gerado para músicas Pop. Apesar de que isto melhore muito a harmonia, acaba consequentemente aumentando a duração da sequência e levantando o desafio da modelação musical de longo prazo. Quanto maior se tornam as peças, mais complexo e oneroso se torna o processo de manter a harmonia constantemente.

Acompanhamentos musicais costumam ser compostos por múltiplos instrumentos/faixas em forma de arranjo, para melhor expressividade. A geração considerando esse contexto pode ser chamada geração multi-faixa. Garantir a harmonia entre notas musicais em diferentes faixas é um problema chave. A representação MIDI criada para possibilitar uma solução do problema se chama MUlti-track MIDI (MuMIDI), que codifica eventos MIDI multi-faixa em uma sequência de tokens. Como o MuMIDI permite a geração de multi-faixas em uma única sequência, a dependência entre as notas musicais em diferentes faixas pode ser melhor capturada e mais informação pode ser aproveitada para melhorar a harmonia.

Para a música ser coerente e interessante, a estrutura de longo prazo — considerando o tamanho e duração das peças — se torna extremamente importante. Isto se torna especialmente relevante visto que o MuMIDI gera múltiplas faixas em uma única sequência, o que como efeito torna consideravelmente mais difícil modelar músicas a longo prazo. Visando uma solução, a duração da sequência foi encurtada em casos específicos, bem como o modelo Transformer-XL de Dai et al. (2019) foi adotado para o codificador e decodificador.

O PopMAG tem assim duas partes essenciais: Representações MuMIDI e um modelo aperfeiçoado sequência-a-sequência. Foram utilizados três diferentes conjuntos de dados de música pop e aplicados tanto testes objetivos quanto subjetivos para avaliação. Os resultados provenientes dos três conjuntos de dados foram promissórios e a aplicação tem um desempenho amplamente superior ao dos sistemas de geração de acompanhamento musical do estado da arte.

Em relação às notas, diferentemente das demais representações, os autores optaram por definir todos os atributos de uma nota (altura, velocidade e duração) em um único símbolo, visando uma simbologia menos verbosa. Então, ao prever múltiplos atributos de uma nota musical, múltiplas matrizes softmax foram adicionadas na saída oculta para gerar atributos correspondentes a esta nota. Ao longo do trabalho de Ren et al. (2020), para acordes, foram consideradas 12 notas raízes (C, C#, D, D#, E, F, F#, G, G#, A, A# e B) e 7 categorias de acorde (maior, menor, diminuto, aumentado, maior7, menor7 e meio_diminuto).

A arquitetura Transformer-XL de Dai et al. (2019) é baseada em Transformer e pode aprender dependências muito mais longas que redes neurais recorrentes (RNNs), por exemplo, além de gerar peças de texto suficientemente coerentes com milhares de

tokens. Por conta disso, foi selecionado para servir de base ao codificador e decodificador.

Para o treinamento, foi utilizada a técnica Teacher Forcing, fornecendo tokens com valor de referência para o decodificador gerar os tokens seguintes. A entropia cruzada entre eles é minimizada visando a otimização do modelo. Para inferência, os tokens alvo são gerados um a um, armazenando a incorporação de compasso/posição atual e a atualizando conforme o conteúdo gerado.

Foram utilizados três conjuntos de dados para avaliar o PopMAG, sendo eles: Lahk MIDI (LMD) apresentado por Raffel (2016), um subconjunto de música pop do FreeMidi e um conjunto de dados de pop chinês MIDI (CPMD). Grande parte destes dados foram gerados pelo usuário, tornando os dados ruidosos demais para serem usados diretamente. Dado este contexto, foram aplicadas etapas de limpeza como extração de melodia, compressão, filtragem e segmentação de dados e reconhecimento de acordes. Assim, cada conjunto de dados foi dividido em 3 partes: 100 amostras para validação, 100 amostras para testes e as demais para treinamento.

Para avaliação subjetiva, foram 15 participantes avaliadores, sendo 5 destes capazes de compreender teoria musical básica. Foram gerados 100 conjuntos de peças, compostas por geradas e valores de referência, para serem avaliados. Cada um é avaliado por todos participantes e o com a melhor harmonia geral deve ser apontado. A média do total de votos de cada conjunto apresenta a pontuação de preferência final.

A avaliação objetiva é composta por métricas inspiradas por Huang e Yang (2020), Yang e Lerch (2020) e Zhu et al. (2018), sendo elas: Precisão do Acorde (CA), que mede se os acordes gerados se encaixam nos condicionais; Perplexidade (PPL), métrica comum em problemas de geração de texto para medir quão bem um modelo se adapta à sequência. A distribuição de algumas características também são usadas para avaliação, comparando com às dos valores de referência. Essas distribuições são calculadas por histogramas suavizados, sendo elas: Altura (P); Velocidade (V), quantificada em 32 níveis de velocidade; Duração (D), também quantificada em 32 atributos de duração e Inter-Onset Interval (IOI), que quantifica os intervalos em 32 classes da mesma forma que a duração da nota. Por fim, a área sobreposta (OA) média das distribuições (DA, onde A pode ser uma de P, V, D e IOI) é calculada para medir a diferença entre a peça musical gerada e a peça musical usada como valor de referência, acordo com Ren et al. (2020).

Observando os resultados promissores do teste aplicado pelos autores para avaliar a harmonia geral e a qualidade das peças musicais gerados, é perceptível que, mesmo existindo uma diferença considerável entre peças compostas por humanos e geradas, cerca de 42%, 38%, 40% em três conjuntos de dados atingiram a qualidade dos valores de referência.

2.3. Considerações

A Tabela 1 faz uma comparação entre os dois trabalhos apresentados em relação às técnicas, conjuntos de dados e categorias de avaliação utilizadas.

Tabela 1. Tabela comparativa das técnicas, conjuntos de dados e categorias de avaliação entre os trabalhos relacionados

Trabalho	Técnica de ML	Conj. de dados	Avaliação
RL-Duet	Apr. por reforço	Bach Chorale	Obj. e Subj.
PopMAG**	Apr. supervisionado	LMD*, CPMD e FreeMidi	Obj. e Subj.

No que lhe concerne, a Tabela 2 mede os dois trabalhos em concernência às métricas utilizadas por eles.

Tabela 2. Tabela comparativa das métricas entre os trabalhos relacionados

*** Métricas baseadas neste conjunto de dados**

**** Métricas da tarefa de piano para outras categorias de acompanhamento**

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
RL-Duet	+0.12	-0.48	-0.09	0.0057	0.042	-	-	-
PopMAG**	-	-	-	-	-	0.58±0.01	0.55±0.01	0.72±0.01

Já a Tabela 3 apresenta métricas diretamente relacionadas aos conjuntos de dados dos trabalhos relacionados.

Tabela 3. Tabela apresentando métricas dos conjuntos de dados dos trabalhos relacionados

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Bach Chorale	3.25	4.57	3.84	-	-

3. Treinamento do modelo e desenvolvimento do protótipo

Um dos principais objetivos deste trabalho é validar a possibilidade de superar, ou ao menos equiparar, as métricas apresentadas nos trabalhos relacionados. Ademais, conforme descrito anteriormente, este estudo visa objetivamente gerar melodias de acompanhamento baseadas em batidas, diferentemente dos demais trabalhos relacionados que aplicaram métodos similares para outros fins. Para isso ser cumprido, a produção musical foi organizada em 7 etapas. São elas: definição do conjunto de dados, escolha, ajustes e treinamento do modelo, avaliação de resultados, ajustes de parâmetros

e desenvolvimento do protótipo. Observando a Figura 1, é possível visualizar o fluxo dessas etapas.

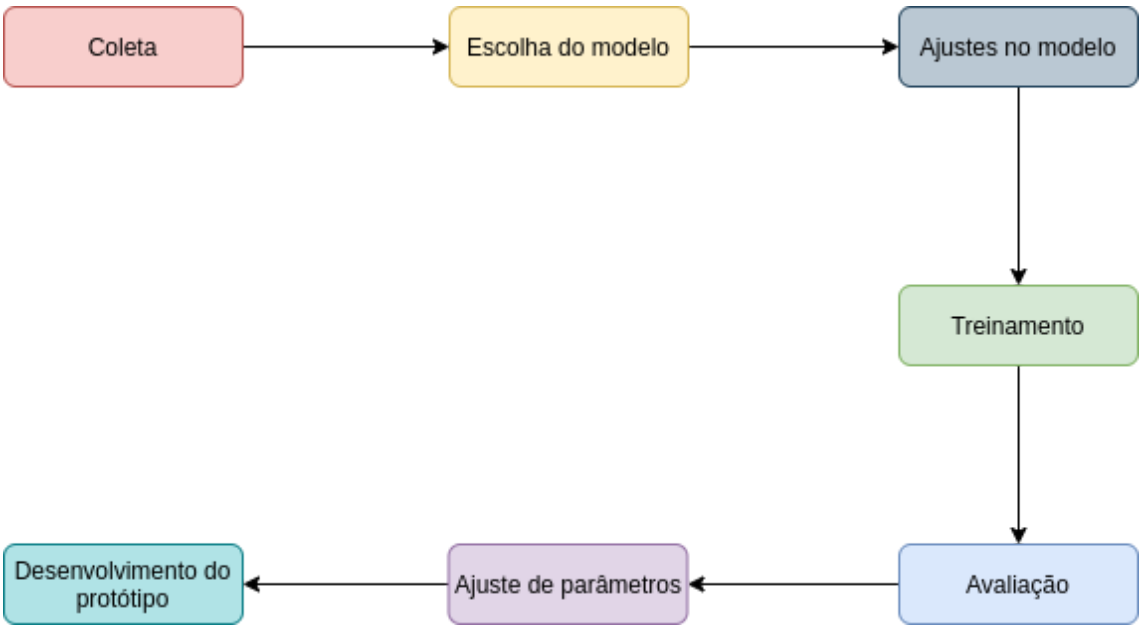


Figura 1. Etapas de concretização do protótipo para geração das melodias de acompanhamento

Na primeira fase, foi realizada uma pesquisa e análise de diversos conjuntos de dados disponíveis online, muitos já utilizados em outros estudos, para selecionar o mais adequado ao escopo deste trabalho. Na etapa de escolha do modelo, foram testadas e analisadas as possíveis opções disponibilizadas pela biblioteca Magenta e selecionada a que mais se encaixava ao caso de uso deste estudo. Seguindo o fluxo, na parte de ajustes no modelo, foram aplicadas mudanças à implementação do modelo e ao processo de treinamento para se adequar ao contexto apresentado. Já na fase de treinamento, o modelo foi treinado com o conjunto de dados previamente determinado e consequentemente avaliado na etapa seguinte, de avaliação. Em sequência, na parte de ajuste de parâmetros, alguns foram adaptados visando melhorar o desempenho. Por fim, foi desenvolvido o protótipo com interface de usuário conforme proposto no objetivo do trabalho.

3.1. Definição do conjunto de dados

Nesta primeira etapa, a coleção de dados que mais se encaixava ao propósito deste estudo foi procurada, analisando e validando diversas coletâneas pré-existentes para aperfeiçoar a seleção com um maior embasamento. No início do processo, foram verificados os próprios conjuntos de dados disponibilizados pelos pesquisadores do projeto Magenta. Tudo considerado código aberto e de livre utilização.

Entre os conjuntos musicais dispostos, haviam dois relacionados à percussão, com atuações variadas de humanos tocando bateria; um relacionado ao piano, com uma

gama de atuações virtuosas e um relacionado às notas musicais, com uma alta densidade de notas tocadas em diferentes instrumentos, cada uma com um tom e timbre únicos.

A coletânea selecionada foi a Groove MIDI Dataset (GMD), constituída pela equipe responsável pela biblioteca Magenta. Como documentado por Gillick et al. (2019), Groove MIDI Dataset possui 13,6 horas distribuídas em 1.150 arquivos MIDI de batidas revisadas, aprofundadas e utilizadas em várias publicações. Uma versão expandida, com 444 horas de áudio, mais metadados e diversidade foi sugerida por Callender, Hawthorne e Engel (2020) com o nome de Expanded Groove MIDI Dataset (E-GMD). Para este trabalho, a versão original, GMD, de Gillick et al. (2019) foi utilizada, em uma opção disponibilizada mais leve, que traz exclusivamente os arquivos MIDI, sem outros tipos de arquivos como WAV.

3.2. Pré-processamento de dados

Um arquivo de registro tensorflow é objeto tensorflow mantido em um arquivo e tratado como um iterador em sequências de notas (RUÉ VILÀ, 2020). O conjunto de dados deve ser convertido com o script convert_dir_to_note_sequences.

A execução cria um arquivo de registro TF no caminho especificado. Dessa forma, o conjunto pode ser carregado pelo arquivo e aplicado no processo de treinamento. Com o TFRecord, é interessante particionar o mesmo para treinamento e processos de avaliação.

Para isso, foi utilizada a classe a RandomPartition da Magenta. O script é extraído do livro de Alexandre Dubreuil (2020).

Com isso, dois tfrecords são criados como eval.tfrecord e train.tfrecord no diretório de saída definido, com uma taxa de avaliação de 0,3. Ou seja, o conjunto de avaliação corresponde a 30% do conjunto de dados.

O train.tfrecord será usado no processo de treinamento, e o eval.tfrecord para avaliar as saídas dadas pelo modelo, visto que elas não foram consideradas no processo de treinamento.

3.3. Escolha do modelo

Para escolha do modelo, foram testados e analisados diversos disponibilizados através da biblioteca Magenta, considerando o contexto deste trabalho e de projetos semelhantes, como outros plugins da Magenta Studio — vistos em Roberts et al. (2019) — ou os apresentados na seção de trabalhos relacionados.

O primeiro modelo cogitado foi o Melody RNN, modelo que utiliza uma rede neural recorrente que aplica modelagem de linguagem à geração de melodias usando memória de curto prazo longo, ou long short-term memory (LSTM). Esse modelo parece imediatamente relevante, dado seu claro objetivo de gerar melodias, que coincide com uma das realizações esperadas por este estudo. Além disso, possui diversas funcionalidades e variações desenvolvidas visando aumentar a diversidade das obras geradas. Por exemplo, o modelo chamado Attention RNN, permite que acesse mais facilmente informações passadas sem ter que armazenar essas informações no estado da célula da RNN, conforme descrito por Waite (2016). Isto permite que o modelo aprenda

mais facilmente dependências a longo prazo, e resulta em melodias com conteúdos mais longos.

Apesar de promissor por seu contexto de geração melódica, o modelo acima não pareceu ideal para gerar melodias como acompanhamento de outros instrumentos, visto que outro modelo mais adequado ao projeto foi encontrado, MusicVAE, descrito abaixo. Ademais, o Melody RNN não foi desenvolvido para receber entradas com sons de outros instrumentos diferentes daquele que fora treinado

O modelo MusicVAE, que utiliza um autocodificador hierárquico recorrente e variável para trabalhar com música, chamou a atenção. Este modelo é particularmente interessante pela sua versatilidade, pois pode ser utilizado em diversos contextos. O MusicVAE aprende um espaço latente de sequências musicais, fornecendo diferentes modos de criação musical interativa, como amostragem aleatória a partir da distribuição prévia; interpolação entre sequências existentes ou manipulação dessas através de vetores de atributos, ou de um modelo de restrição latente, segundo Roberts et al. (2018).

Com essas informações, foi possível perceber que esse modelo poderia se alinhar com o desejado. Estudando o MusicVAE mais profundamente, foi encontrada uma variação desse modelo, chamada GrooVAE. Ela visa gerar e controlar atuações expressivas de bateria, facilitando variados desafios de produção. Foi utilizada com o conjunto de dados de bateria da biblioteca Magenta para treinar o modelo que fomenta o plugin Druminfy.

Para validar a possibilidade do uso do MusicVAE, foram aplicadas sessões de treinamento com o conjunto de dados definido anteriormente. Houveram alguns resultados errôneos, mas aparentou que, com alguns ajustes, o modelo poderia trazer os valores esperados.

3.4. Ajustes no modelo

Para adequar o MusicVAE ao contexto deste estudo, foram necessárias algumas mudanças no código-fonte da sua implementação na biblioteca Magenta. Durante a análise para determinação do modelo, foi proposto que o melhor caminho para seguir seria basear-se naquele treinado para o plugin Drumify.

A configuração do modelo mais simples relacionado ao plugin é definido como groovae_2bar_tap_fixed_velocity. Ele converte 2 compassos de um padrão “tap” de velocidade constante em uma peça de bateria. É definido com um codificador bidirecional LSTM e um decodificador LSTM de batidas. Para hiperparâmetros, possui tamanho do lote; tamanho máximo da sequência em compassos; medida do vetor latente z; número de unidades por camada da RNN do codificador; número de unidades por camada da RNN do decodificador; peso máximo do custo Kullback–Leibler (KL) e bits para excluir da perda KL por dimensão. Utiliza também um conversor de dados para batidas, presente no arquivo data.py e responsável por transfazer bidirecionalmente representações como batida (hit)/velocidade (velocity)/compensação (offset). Como Gillick (2019) pontua, a batida representa se houve ocorrência de determinada nota na sequência, e velocidade representa a força da batida de bateria. Compensação nesse caso

se refere à distância relativa e em que direção se encontra o tempo de cada nota em relação à 16.ª nota mais próxima. (GILLICK, 2019)

Para alcançar os objetivos deste estudo com esforço hábil, foi priorizada a reutilização das pré-definições presentes na biblioteca. Analisando o conversor de dados e entendendo como a técnica chamada de "squash" — transformar batidas em um "tap" constante — funciona, foi possível identificar potenciais mudanças que auxiliariam a nortear o projeto. Como dito anteriormente, cada timestep contém uma representação de bateria, que pode ser uma de 9 categorias definidas no projeto. Como este trabalho visa gerar melodias de piano, essas categorias foram adequadas para uma simples listagem de notas deste instrumento, variando das alturas 0 ao 127 na notação MIDI, como pode ser constatado na Figura 2.

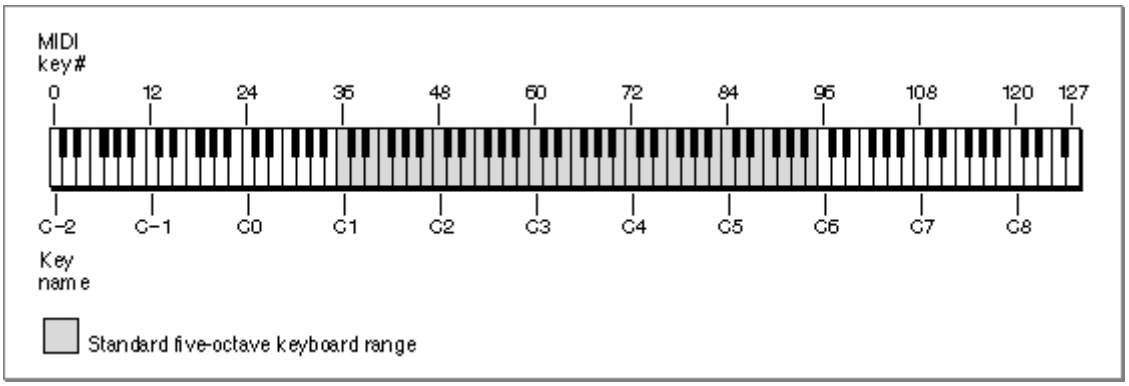


Figura 2. Valores de notas MIDI e teclas de piano correspondentes

(APPLE COMPUTER INC, 1996)

Ademais, ao converter de volta dos tensores e iterar através dos instrumentos para criar uma nota da sequência, as informações da nota tiveram seu instrumento MIDI alterado de 9 para 1, ou seja, de bateria para piano. A propriedade is_drum foi definida como falsa, que determina se a nota é de uma bateria ou não.

3.5. Treinamento

Após os ajustes compreendidos e aplicados, uma nova configuração foi criada e treinada para este estudo.

Para treinar um modelo com dados próprios, primeiro é necessário converter um conjunto de dados MIDI em um TFRecord de NoteSequences, ou seja, um arquivo compatível com variações do TensorFlow para armazenar uma sequência de registros binários, representando numericamente as notas musicais. Esta representação de dados armazena aspectos fundamentais das sequências de anotações (tempo, alturas, instrumentos, etc.) assim como metadados adicionais (etiquetas de seção, informações de acorde, etc.). É serializável e pode ser acessada e modificada através de várias linguagens. (ROBERTS, HAWTHORNE e SIMON, 2018)

Após isso, com o modelo ajustado na biblioteca local e o conjunto de dados pronto para treinamento, foram executados os comandos necessários para iniciar. Os modelos foram treinados para testagem, com hiperparâmetros adaptados às

especificidades da máquina física usada no estudo, que possui limitações em aspectos como espaço, memória e GPU. O tamanho do lote foi diminuído de 512 para 32 e a taxa de aprendizado de 0,001 para 0,005. Cada modelo passou entre 3 e 12 horas em treino.

3.6. Avaliação

Nesta etapa, foi executada uma análise do modelo considerando as métricas do capítulo 2.5 e outras com fins mais generalistas, visando verificar e potencialmente aprimorar o desempenho, a eficácia e a eficiência. Também foram utilizadas algumas ferramentas para isso, como o Tensorboard e o MGEval.

3.6.1 Tensorboard

O Tensorboard reúne ferramentas para visualização e experimentação de processos utilizados no TensorFlow. Dentro das suas capacidades, existem ações como rastreamento e visualização de métricas, exibição de dados, visualização de histogramas de parâmetros, conforme levantado por Rué Vilà (2020).

Ao longo do treinamento, o modelo salva rotineiramente um checkpoint, ou ponto de verificação, que guarda informações do estado atual dos parâmetros. Finalizando o treinamento, é possível visualizar a evolução em qualquer navegador com o uso do Tensorboard.

O código do modelo possui elementos do Tensorboard definidos, porém ignora o mapa métrico que corresponde a cada decodificador específico, ou seja, está utilizando uma métrica generalizada, e não parece ser o melhor rastreamento para todos os casos. O modelo está sempre utilizando o resumo geral para perdas (RUÉ VILÀ, 2020).

Para o modelo GrooVAE, o mapa métrico é definido em um método chamado flat_reconstruction_loss no GrooveLSTMDecoder. De todo modo, no modelo padrão, esses valores são ignorados e os escalares gerais são usados.

A visualização obtida com o Tensorboard é similar às apresentadas nas Figuras 3 e 4:

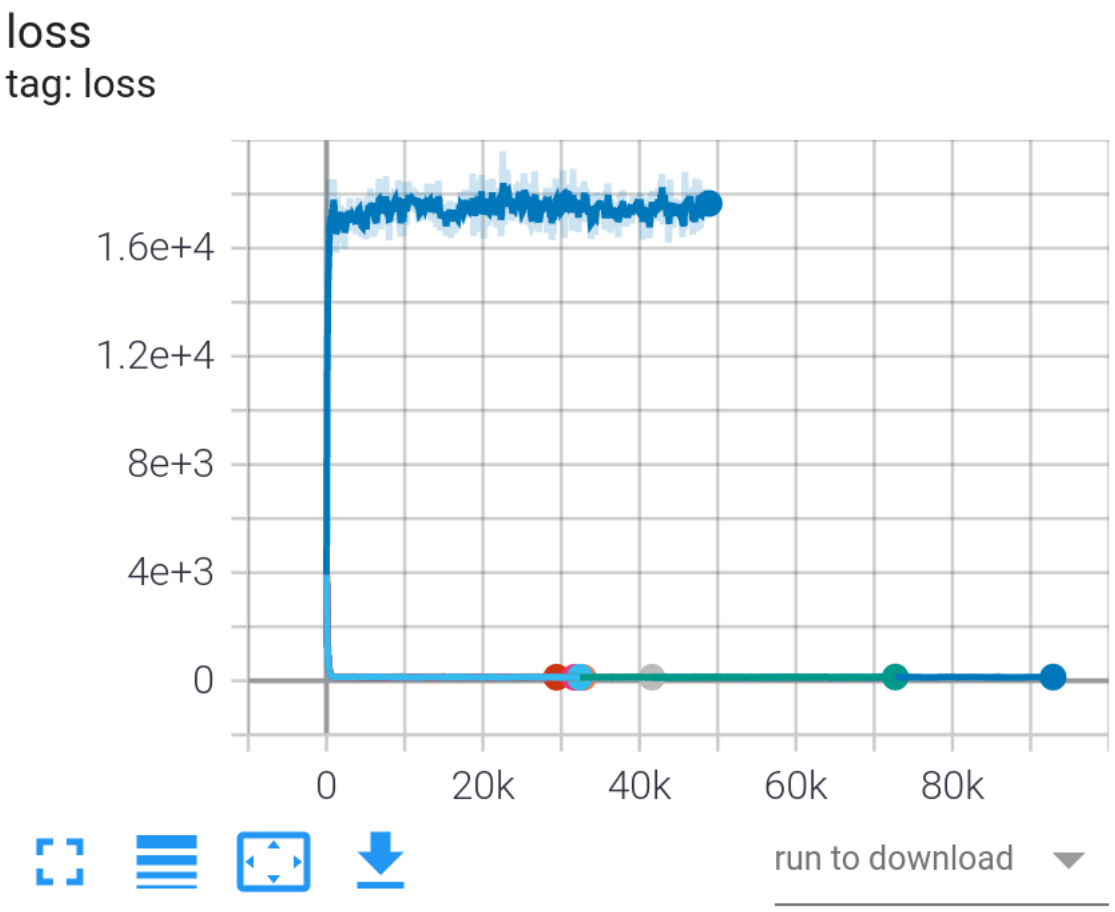


Figura 3. Gráfico de perdas geral

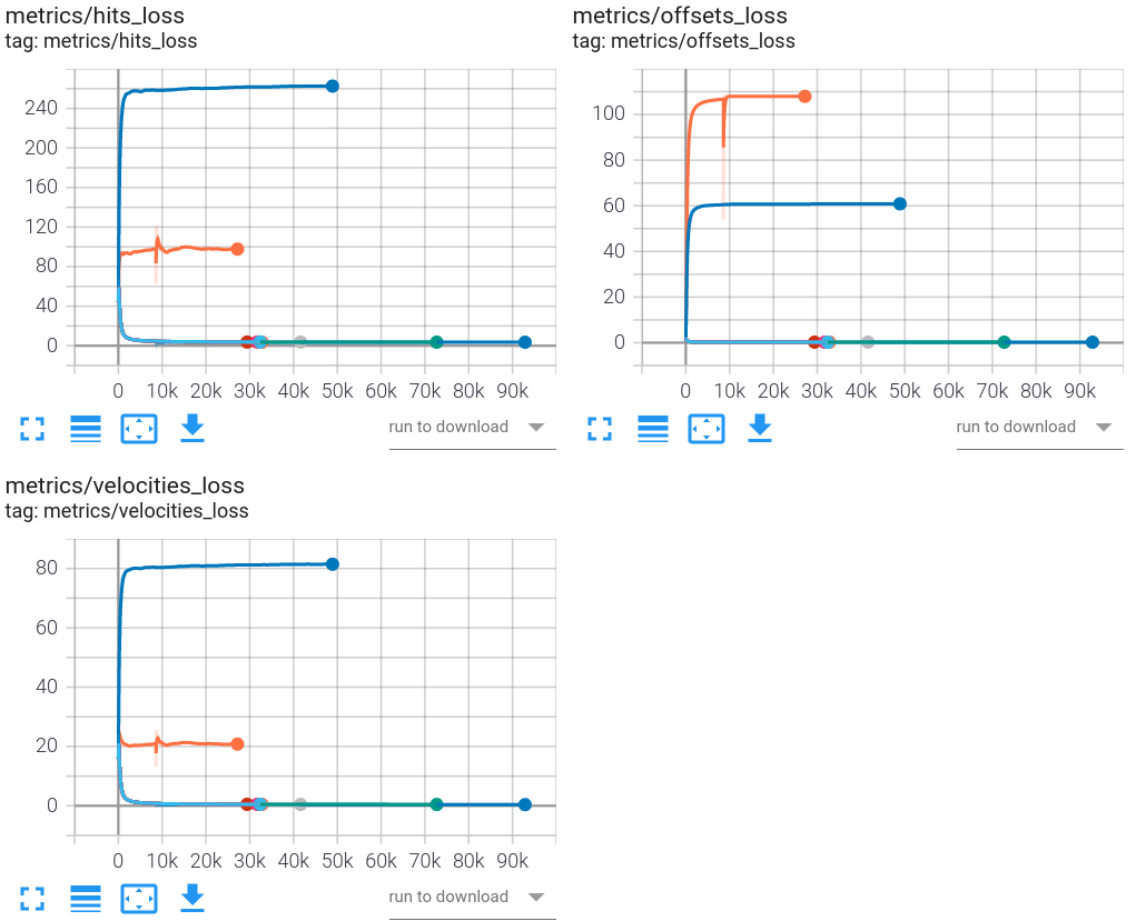


Figura 4. Ampliação das métricas que acompanham as perdas de batidas, compensações e velocidades

Na Figura 3, é possível visualizar a perda global, ou seja, a soma das três perdas específicas. Já na área inferior, estão dispostas as perdas específicas. No eixo y, é identificável a quantia da perda calculada, e o eixo x representa o passo no otimizador. Sobre a perda, vemos uma faixa diferente na perda de batidas do que nas outras duas. Isto porque a perda de batidas é uma perda de entropia cruzada, as compensações e velocidades são erros quadráticos médios (RUÉ VILÀ, 2020). Dado o contexto geracional do estudo, aparenta-se que uma perda com erro quadrático médio seria mais otimizada que a entropia cruzada, visando esse valor para melhoria na perda de batidas.

3.6.2 MGEval

MGEval, assim como o Tensorboard, é uma caixa de ferramentas de código aberto. Ela dispõe de métodos de avaliação, análise e visualização para sistemas musicais generativos, conforme pontuado por Yang e Lerch (2020). Seu objetivo é fornecer um sistema de avaliação objetivo para ter uma melhor confiabilidade, validade e reprodutibilidade na geração musical.

MGEval é responsável por calcular métricas e outras adicionais, propondo um conjunto de medidas objetivas musicalmente informadas para obter análises que impulsionam ainda mais séries de estratégias de avaliação. As métricas são

categorizadas como absolutas ou relativas. Respectivamente, fornecendo intuições sobre as propriedades do conjunto de dados gerado ou coletado, e comprando dois conjuntos de dados de treinamento e gerado.

Apesar da biblioteca possuir todo esse ecossistema, ainda precisaram ser feitos alguns ajustes para o funcionamento pleno, especialmente no Python 3. A atualização de algumas dessas bibliotecas, como a python-midi, foi realizada, bem como codificação extra para contornos de casos específicos, cálculos e gráficos adicionais. Bons exemplos destes cálculos são a distância de Wasserstein entre o histograma da classe de altura dos arquivos MIDI gerados e de teste, e entre o histograma de comprimento de nota dos mesmos arquivos. Os gráficos apresentam a medição da diferença de distâncias intra-set e inter-set por KLD e OA.

Com isso, o arquivo __main__.py é executado e com ele todas as funcionalidades da MGEval. Após a finalização do processo, os resultados podem ser visualizados pelo terminal ou, de uma forma mais estruturada, em um JSON com as informações.

Para cada feature, é calculada a média e o desvio padrão dos arquivos MIDI gerados e de teste, bem como duas métricas considerando as PDFs das distâncias intra-set do conjunto de treinamento e inter-set entre os conjuntos de dados de treinamento e gerado: OA e KLD. Adicionalmente, nas features que retornam um histograma, calcula a distância de Wasserstein entre a média dos arquivos MIDI gerados e de teste. A menor diferença, ou distância, sugere uma melhor imitação de estilo do conjunto de dados. (JIANG, 2020)

Tabela 4. Tabela com métricas baseadas nos arquivos gerados e de teste deste trabalho

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
Este trabalho	+1.90	+1.13	+0.015	0.083	0.015	0.0001±0.016	0.27±0.01	0.83±0.02

Tabela 5. Tabela com métricas do conjunto de dados

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Este trabalho	3.15	7.71	0.13	-	-

Para as primeiras 5 métricas, esperamos um valor mais próximo possível de 0, visto que demonstra a distância dos valores de referência. Em contraste, para as distribuições, esperamos um valor mais próximo possível de 1, considerando que representa a área sobreposta entre arquivos de teste e aqueles gerados pelo modelo. Estes valores demonstram, acima de tudo, a similaridade entre os conjuntos de dados fornecidos para a ferramenta, conforme pontuado por Yang e Lerch (2020), neste trabalho sendo o conjunto de teste, e o de arquivos gerados.

3.7 Otimização do modelo

Primeiramente, foram realizados treinamentos com foco apenas no ajuste da taxa de aprendizagem do modelo. Esta taxa informa a grandiosidade das atualizações em cada etapa do otimizador.

Um diretório pai é determinado para salvar os pontos de verificação de cada treinamento em pastas dedicadas, simplificando assim a visualização dos resultados usando o Tensorboard.

Foram testadas taxas de aprendizado com valor 0.1, 0.01, 0.001, 0.0005, 0.0008, 0.002, 0.003, 0.004 e 0.005, assim como o trabalho de Rué Vilà (2020). Finalizando o treinamento para cada taxa, é possível visualizar e comparar os modelos pelo navegador. Essa comparação torna possível a determinação do valor da taxa de aprendizagem que gera o menor custo.

Analisando e ranqueando os resultados, podemos concluir que, de maneira geral, a taxa de aprendizado com valor 0.005 apresentou os resultados mais equilibrados e satisfatórios entre os três tipos de perda.

Na sequência, foram comparados diversos tamanhos de rede. O valor padrão vem definido na seguinte estrutura: enc_rnn=[512], dec_rnn=[256,256] e z_size=256.

Ainda seguindo os experimentos de Rué Vilà (2020), manteve-se a estrutura de enc_rnn=[X], dec_rnn=[X/2,X/2], z_size=X/2, e testaram-se os seguintes valores para X: 256, 384, 512, 768 e 1024. Com a taxa de aprendizagem fixada em 0,005.

Tabela 6. Tabela com métricas dos modelos com diferentes tamanhos de rede

Tamanho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
256	+2.40	+0.51	-0.025	0.083	0.015	0.0003±0.35	0.27±0.01	0.63±0.06
384	+2.15	**+0.31**	**-0.009**	0.083	0.015	0.0001±0.08	0.27±0.01	0.73±0.07
512	+3.30	+1.09	-0.021	0.083	0.015	**0.015±0.14**	0.27±0.01	0.69±0.04
768	**+1.90**	+1.13	+0.015	0.083	0.015	0.0001±0.016	0.27±0.01	**0.83±0.02**
1024	+2.45	+1.96	-0.033	0.083	0.015	0.0018±0.078	0.27±0.01	0.53±0.04

Como pode ser concluído através da Tabela 6, o codificador que teve o melhor desempenho foi o de tamanho de rede = [768], decodificador = [384,384] e z_size = 384.

3.8 Desenvolvimento do protótipo

A etapa final foi executada visando alcançar um dos principais objetivos propostos neste estudo, implementar um protótipo com interface para tornar hábil e conveniente o uso do modelo em seu contexto de plugin, assim como o Drumify. Aqui, um servidor simples com Node.js foi implementado para servir os arquivos estáticos do modelo. A parte do cliente foi desenvolvida com Vue.js na versão 3 como arcabouço principal, bem como Stylus para estilização, Magenta.js (ROBERTS, HAWTHORNE e SIMON, 2018) para consumir e aplicar o modelo, e outras bibliotecas ou pacotes. Esta etapa teve grande foco no visual e no interativo, o que considera a preocupação com a usabilidade da aplicação visando aprimorar a experiência do usuário final. Também houve grande inspiração dos plugins já existentes da biblioteca Magenta, como o próprio Drumify. Desta forma, a versão deste estudo foi nomeada Melodify.

Antes da implementação dos projetos, foi necessário converter os pesos dos checkpoints do modelo treinado para o formato utilizado pela Magenta.js, diferente daquele fornecido como saída dos treinamentos da biblioteca python Magenta, utilizando o script checkpoint_converter, disponibilizado na versão JavaScript (JS) da biblioteca.

Além disso, também foi preciso especificar determinados parâmetros do modelo em um arquivo JSON, especificado como config.json no mesmo diretório que os checkpoints convertidos. Este arquivo de configuração contém todas as informações necessárias (além dos pesos) para instanciar e executar o modelo: a categoria de modelo

e a especificação do conversor de dados mais a codificação de acordes opcional, entradas auxiliares, entre outras opções.

Seguindo, foi implementado o servidor responsável por servir os arquivos convertidos e criados nos passos anteriores para a aplicação no lado do cliente. As principais tecnologias utilizadas foram Node.js e Express, arcabouço para rodar JS como back-end e um gerenciador de rotas para Node.js, respectivamente. Outros pacotes também foram utilizados, como um encarregado de manejar as requisições e prevenir erros Cross-Origin, ou seja, suporta que materiais privados sejam recuperados por um domínio diferente do domínio ao qual pertence o material que será recuperado. Esta aplicação serve na porta 3000 da máquina local e possui apenas a rota /static, capaz de retornar os arquivos dos checkpoints convertidos do modelo, que devem estar presentes dentro de /models/converted no diretório do projeto.

Como mencionado anteriormente, houve grande inspiração nos plugins já existentes na Magenta Studio (ROBERTS et al., 2019). A interface apresentada simplifica ao máximo o uso de um modelo que grande parte dos potenciais usuários alvo, como músicos e afins, poderiam considerar complexo demais de outra forma, já que não necessariamente possuem grande aptidão tecnológica. Desta forma, é possível apenas enviar um arquivo MIDI de qualquer instrumento e receber uma atuação de bateria para acompanhamento. Porém, diferentemente do protótipo apresentado neste trabalho, os plugins da Magenta Studio são maduros e robustos, além de utilizarem React ao invés de Vue como arcabouço principal e Electron, para rodarem como aplicações nativas desktop ao invés de diretamente no browser.

Dessa forma, foi criada a parte do cliente do Melodify. O protótipo utiliza Vue na versão 3, Stylus e Magenta.js (ROBERTS, HAWTHORNE e SIMON, 2018). Primeiramente, foi criada uma classe para utilizar a Magenta.js e consumir o modelo que será servido através do servidor descrito acima. Essa classe também é responsável por receber a entrada de bateria em formato MIDI enviada pelo usuário e, posteriormente, gerar a melodia para acompanhamento. Além da classe, foram criados componentes de botão, entrada de arquivo, rótulo e seleção em intervalo. A tela principal é implementada com ditos componentes.

Após enviar o arquivo MIDI contendo a entrada com uma atuação de bateria e selecionando a temperatura, o botão Generate fica habilitado e o usuário pode clicar para gerar suas melodias de acompanhamento. O protótipo está apresentado na Figura 5.

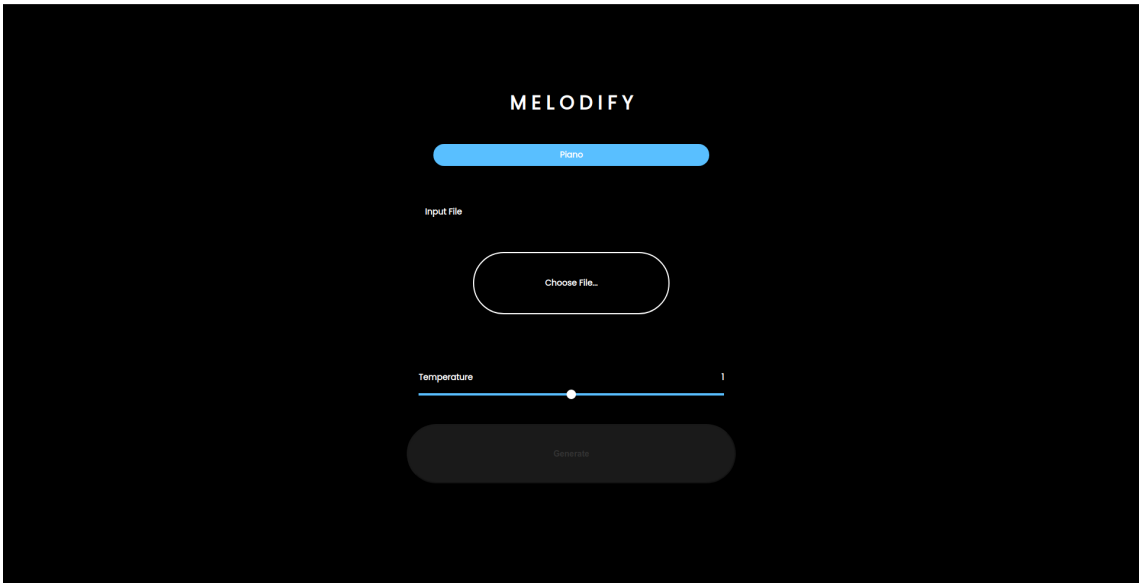


Figura 5. Interface do Melodify rodando no browser

4. Comparação com os trabalhos relacionados

Comparando com os trabalhos relacionados, é perceptível que, similarmente ao trabalho 1, de Jiang et al. (2020) (1) e 2, de Ren et al. (2020) (2), neste trabalho foi usado um conjunto de dados existente, ou seja, não executou um processo de coleta de dados, como apresentado na Tabela 7.

Tabela 7. Tabela comparativa das técnicas, conjuntos de dados e categoria de avaliação entre os artigos coletados e este trabalho

Trabalho (n.º)	Técnica de ML	Conj. de dados	Avaliação
RL-Duet (1)	Apr. por reforço	Bach Chorale	Obj. e Subj.
PopMAG (2)**	Apr. supervisionado	LMD*, CPMD e FreeMidi	Obj. e Subj.
Este trabalho	Apr. não supervisionado	Groove MIDI Dataset	Majoritariamente Obj.

Uma diferença notável fica por parte da técnica de ML. O trabalho 1 utiliza aprendizado por reforço, enquanto o segundo adota o aprendizado supervisionado, e este aplica aprendizado não supervisionado. Outra diferença é a avaliação. O trabalho 1 e 2 recorreram a técnicas objetivas e subjetivas para avaliar seus resultados, enquanto este efetuou uma avaliação majoritariamente objetiva. O trabalho 1 utilizou métricas objetivas que também foram utilizadas neste projeto, mas também aplicou um extenso estudo com usuários, que considerava diferentes níveis de conhecimento musical ao avaliar os resultados. O trabalho 2 também aplicou métricas que estão aqui apresentadas, além de um estudo humano com 15 participantes, tendo 5 que

compreendem a teoria básica da música. Este trabalho, por conta de tempo hábil e escopo, tem uma avaliação majoritariamente objetiva. A parte subjetiva pode ser dada pelas opiniões pessoais.

Levando em conta as métricas, podem ser comparadas e estão dispostas na Tabela 8. O PC/Bar apresentou uma diferença de +1.90 em relação ao conjunto de dados de teste; enquanto o PI e o IOI mostraram diferenças de +1.13 e +0.015, respectivamente. As distâncias de Wasserstein entre a música gerada e o conjunto de dados de teste para PCH e NLH foram definidas como 0.083 e 0.015, também respectivamente. As distribuições de altura (DP), duração (DD) e intervalo entre notas (DIOI) são 0.0001±0.016, 0.27±0.01 e 0.83±0.02, respectivamente.

Tabela 8. Tabela comparativa das métricas entre os artigos coletados e este trabalho

*** Métricas baseadas neste conjunto de dados**

**** Métricas da tarefa de piano para outras categorias de acompanhamento**

Trabalho	PC/bar	PI	IOI	PCH	NLH	D_P	D_D	D_{IOI}
RL-Duet	**+0.12**	**-0.48**	-0.09	**0.0057**	0.042	-	-	-
PopMAG**	-	-	-	-	-	**0.58±0.01**	**0.55±0.01**	0.72±0.01
Este trabalho	+1.90	+1.13	**+0.015**	0.083	**0.015**	0.0001±0.016	0.27±0.01	**0.83±0.02**

Tabela 9. Tabela comparativa entre os conjuntos de dados analisados

Conj. de dados	PC/bar	PI	IOI	PCH	NLH
Bach Chorale	3.25	4.57	3.84	-	-
Este trabalho	3.15	7.71	0.13	-	-

Dentro das oito métricas apresentadas, o projeto apresentou valores promissores em três delas quando comparado aos demais trabalhos. Quando comparado ao trabalho 1, apresentou uma diferença ao conjunto de dados de teste inferior na métrica IOI, apresentando uma maior semelhança ao seu respectivo conjunto de dados de teste, mesmo que ligeiramente em sua maioria. Adicionalmente, demonstra uma distância de Wasserstein inferior considerando o NLH.

Já em comparação com o trabalho 2, apresenta métricas favoráveis apenas em uma das três comparações. Na distribuição de intervalo entre notas (DIOI), possui um valor sutilmente superior ao apresentado no outro projeto, indicando uma leve vantagem no contexto desta feature.

O projeto obteve resultados relevantes, mas deve ser notada a potencial influência do uso de um conjunto de dados com sons de bateria para o treinamento e teste nas métricas. Algumas das bibliotecas utilizadas retornavam valores zerados em algumas métricas relacionadas à altura dos arquivos, mencionado anteriormente.

5. Conclusão

Este projeto teve a intenção de considerar outras pesquisas e modelos que aplicaram os conceitos de acompanhamento e geração musical, porém focado em batidas de bateria e gerar melodias de piano para o acompanhamento, especificando o modelo elaborado.

Consequentemente, é possível concluir que o projeto atingiu seu objetivo geral. Os objetivos específicos, similarmente, foram cumpridos. O estado da arte em Machine Learning para geração musical foi reconhecido por meio dos trabalhos relacionados, que foram utilizados para assimilar os métodos e sugerir o modelo de geração do projeto deste trabalho. Além disso, diversas métricas foram aplicadas, analisadas e os resultados finais foram levantados e comparados aos trabalhos relacionados.

No decorrer do desenvolvimento do modelo, um tanto de adversidades foram encontradas. A primeira delas foi a falta de compatibilidade de algumas das ferramentas com a nova versão padrão do Python, 3. Além disso, diversas mudanças na biblioteca original e nas ferramentas de métricas precisam ser realizadas para o pleno funcionamento. A falta de estudos com algumas das ferramentas utilizadas também foi um fator adverso. É um tema bastante embrionário, porém extremamente promissor e brilhante.

A comparação com os trabalhos relacionados foi complexa, visto que muitos trabalhos apresentam suas métricas de maneira diversa, ou métricas completamente diferentes. Isto inclusive é um assunto abordado em diversos dos estudos utilizados para referência e anteriormente neste trabalho. Apesar do estudo de Yang e Lerch (2020) definir um bom padrão, é recente e ainda não foi amplamente difundido e adotado. Nos estudos em que é utilizado, pode ser parcialmente aplicado ou ter seus valores representados de formas variadas.

Deste modo, baseado na execução deste trabalho ficam evidentes algumas oportunidades de futuras evoluções que podem ser elaboradas e que estão diretamente relacionadas ao trabalho atual. Estes pontos estão brevemente descritos abaixo.

Visando obter resultados mais confiáveis e precisos, é recomendado a avaliação quantitativa de outros tipos de modelo. Neste projeto, foi considerado aquilo já desenvolvido pelos responsáveis pela biblioteca Magenta, baseando-se nos seus experimentos e resultados para tomar decisões que poderiam levar em conta uma análise própria e mais profunda. Isto poderia ser alcançado treinando outros modelos e comparando métricas semelhantes às apresentadas neste trabalho.

Apesar da interface gráfica ter sido desenvolvida e conectada com o modelo treinado, ela ainda pode ser aprimorada . Isto pode ter estar relacionado a biblioteca Magenta.js e como ela interpreta o modelo customizado desenvolvido ao longo deste trabalho, mas uma avaliação mais assertiva deve ser realizada para encontrar de fato os problemas e possíveis soluções.

Ademais, o modelo pode ser aprimorado e expandido. Pode ser estudada uma maneira de receber e gerar sons de diversos outros instrumentos, tendo assim uma maior variedade musical e aplicabilidade prática. O modelo desenvolvido neste trabalho foca em receber batidas de bateria e gerar melodias de piano. Isto provavelmente implicaria em ainda mais mudanças na biblioteca Magenta e no processo de treinamento.

Assim como o ponto anterior, uma integração do modelo com ferramentas musicais poderia aumentar drasticamente a aplicabilidade e uso prático. Isto já foi realizado com outros modelos criados pela equipe responsável pela biblioteca Magenta na ferramenta Ableton, o que pode facilitar o aprimoramento do modelo desenvolvido neste trabalho. Outra ferramenta semelhante ao Ableton que poderia ser utilizada para integração é o Fruity Loops, ou, como é mais comumente conhecido, FL Studio.

O conceito de estrutura de longo prazo, mencionado ao longo deste projeto, também pode ser considerado com mais importância em futuras evoluções. Como o modelo foi treinado para gerar apenas dois compassos por arquivo MIDI, a estrutura de longo prazo não pôde ser avaliada com confiabilidade nos resultados. Um bom caminho pode envolver o aumento no número dos compassos considerados ao treinar o modelo geracional.

Também poderia ser de interesse aplicar uma avaliação subjetiva com seres humanos, com conhecimento musical e/ou não, para validar os resultados do estudo além das métricas objetivas e olhar pessoal potencialmente enviesado. Esse tipo de

avaliação é comumente utilizada em outros estudos criativos voltados ao campo musical.

References

KASHYAP, Ravi. The universal language: mathematics or music?. Journal for Multicultural Education, 2021.

ONGSULEE, Pariwat. Artificial intelligence, machine learning and deep learning. In: 2017 15th international conference on ICT and knowledge engineering (ICT&KE). IEEE, 2017. p. 1-6.

HUANG, Yu-Siang; CHOU, Szu-Yu; YANG, Yi-Hsuan. Pop music highlighter: Marking the emotion keypoints. arXiv preprint arXiv:1802.10495, 2018.

LU, Chien-Yu et al. Play as you like: Timbre-enhanced multi-modal music style transfer. Proceedings of the AAAI Conference on Artificial Intelligence. 2019. p. 1061-1068.

ROBERTS, Adam et al. A hierarchical latent vector model for learning long-term structure in music. In: International conference on machine learning. PMLR, 2018. p. 4364-4373.

JIANG, Nan et al. RL-Duet: Online Music Accompaniment Generation Using Deep Reinforcement Learning. Proceedings of the AAAI Conference on Artificial Intelligence. 2020. p. 710-718.

CUTHBERT, Michael Scott; ARIZA, Christopher. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

HADJERES, Gaëtan; PACHET, François; NIELSEN, Frank. Deepbach: a steerable model for bach chorales generation. International Conference on Machine Learning. PMLR, 2017. p. 1362-1371.

BRIOT, Jean-Pierre; HADJERES, Gaëtan; PACHET, François-David. Deep learning techniques for music generation--a survey. arXiv preprint arXiv:1709.01620, 2017.

YANG, Li-Chia; LERCH, Alexander. On the evaluation of generative models in music. Neural Computing and Applications, v. 32, n. 9, p. 4773-4784, 2020.

REN, Yi et al. Popmag: Pop music accompaniment generation. Proceedings of the 28th ACM International Conference on Multimedia. 2020. p. 1198-1206.

DAI, Zihang et al. Transformer-xl: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860, 2019.

RAFFEL, Colin. Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. 2016. Tese de Doutorado. Columbia University.

HUANG, Cheng-Zhi Anna et al. Music transformer: Generating music with long-term structure. International Conference on Learning Representations. 2018.

HUANG, Yu-Siang; YANG, Yi-Hsuan. Pop music transformer: Generating music with rhythm and harmony. arXiv preprint arXiv:2002.00212, 2020.

ZHU, Hongyuan et al. Xiaoice band: A melody and arrangement generation framework for pop music. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018. p. 2837-2846.

ROBERTS, Adam et al. Magenta studio: Augmenting creativity with deep learning in ableton live. 2019.

WAITE, Elliot. Generating Long-Term Structure in Songs and Stories. 2016. Disponível em: https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn. Acesso em: 19 set. 2021.

RUÉ VILÀ, Aleix. Drums generation from a tapped pattern. 2020. Trabalho de Conclusão de Curso. Universitat Politècnica de Catalunya.

DUBREUIL, Alexandre. Hands-On Music Generation with Magenta. Packt Publishing, 2020. Disponível em: https://github.com/PacktPublishing/hands-on-music-generation-with-magenta. Acesso em: 16 jan. 2022.

GILLICK, Jon et al. Learning to groove with inverse sequence transformations. In: International Conference on Machine Learning. PMLR, 2019. p. 2269-2279.

ROBERTS, Adam; HAWTHORNE, Curtis; SIMON, Ian. Magenta.js: a JavaScript API for augmenting creativity with deep learning. 2018.

CALLENDER, Lee; HAWTHORNE, Curtis; ENGEL, Jesse. Improving perceptual quality of drum transcription with the expanded groove midi dataset. arXiv preprint arXiv:2004.00188, 2020.

CROCKFORD, Douglas. The application/json media type for javascript object notation (json). 2006.