UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES
CAMPUS PATOS DE MINAS

ELISEU ELIAS CÂNDIDO MOREIRA

APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA O RECONHECIMENTO DE ELEMENTOS DE PARTITURAS MUSICAIS

Patos de Minas - MG
2024

ELISEU ELIAS CÂNDIDO MOREIRA

APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA O RECONHECIMENTO DE ELEMENTOS DE PARTITURAS MUSICAIS

Trabalho de Conclusão de Curso apresentado à banca examinadora como requisito da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, *Campus* Patos de Minas.
Orientadora: Profª. Drª. Karine Barbosa Carbonaro
Coorientador: Me. Daniel de Oliveira Ferreira

Patos de Minas - MG

2024

APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA O RECONHECIMENTO DE ELEMENTOS DE PARTITURAS MUSICAIS

Trabalho de Conclusão de Curso aprovado para a obtenção do título de Bacharel em Engenharia Eletrônica e de Telecomunicações da Universidade Federal de Uberlândia – *Campus* Patos de Minas (MG) pela banca examinadora composta por:

Aprovado em 29 de janeiro de 2024.

Profª. Drª. Karine Barbosa Carbonaro, UFU/MG
(Orientadora)

Me. Daniel de Oliveira Ferreira, UFU/MG
(Coorientador)

Prof. Dr. Rafael Augusto da Silva, UFU/MG
(Membro)

Prof. Dr. Renan Alves dos Santos, UFU/MG
(Membro)

"Não acumulem tesouros sobre a terra, onde as traças e a ferrugem corroem e onde ladrões escavam e roubam; mas ajuntem tesouros no céu, onde as traças e a ferrugem não corroem, e onde ladrões não escavam, nem roubam. Porque, onde estiver o seu tesouro, aí estará também o seu coração". Mateus 6:19-21.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à Deus pelo dom da vida que me deu, pois somente após a morte de Seu único Filho na cruz, meus pecados foram perdoados, e, portanto, fui libertado do império das trevas e transportado para o Reino do seu Filho (Cl. 1.13). Pela Sua livre graça e misericórdia que fluem do Seu amor incondicional, tenho sido sustentado até o presente momento e os que ainda virão.

Agradeço à minha família, que apesar dos pesares, sempre esteve comigo e me apoiou em tudo, desde o início até este momento de grande conquista que é me tornar um Engenheiro Eletrônico e de Telecomunicações.

Sou grato também à minha segunda família, a família da fé, que é a 3 Igreja Presbiteriana de Patos de Minas, pelas orações, pelo apoio, pelo cuidado e pelo acompanhamento em todos os momentos que necessitei.

Finalmente, agradeço aos meus professores e amigos que muito me auxiliaram, desde o ciclo fundamental até o fim desta jornada ao longo destes seis anos, em meio ao caos total devido à pandemia de COVID-19. Durante todo este tempo consegui evoluir, nem que seja um pouquinho, como ser humano tanto na área acadêmica como na área profissional. Foi por conta de cada um deles, pelas motivações, paciência e ajuda que hoje pude entender que a vida acadêmica vai além de simplesmente estudar. Ademais, qualquer palavra acrescentada ainda seria insuficiente para descrever a minha gratidão.

Finalizo com um último versículo das Escrituras, palavra que me sustenta todos os dias, compartilhando o pensamento do apóstolo Paulo em 2Timótio 4.7: “Combati o bom combate, completei a corrida, guardei a fé”, o fim desta etapa significa apenas o início de uma nova jornada.

RESUMO

A importância da música para humanidade é notória, uma vez que influencia diversos aspectos das sociedades há séculos. Além do valor cultural, pode contribuir no desenvolvimento cognitivo e está relacionada a um mercado de grande importância financeira. No entanto, a diversidade, o volume e a complexidade dos registros musicais tornam a manutenção e manipulação de partituras um desafio. Para digitalizá-las, processos manuais podem consumir tempo e estar sujeitos a erros. Assim, a utilização de métodos de reconhecimento automático de partituras é um objetivo pertinente. Neste contexto, o presente trabalho explora a aplicação de redes neurais na classificação de elementos de partituras musicais. Para executar tal tarefa são utilizadas redes *perceptron* multicamadas (MLP) e redes neurais convolucionais (CNN). Diversas configurações foram testadas para avaliar a influência de parâmetros como o número de amostras por *batch*, número de épocas de treinamento, dimensões do *kernel*, bem como número de camadas, sejam convolucionais, densas, de *dropout* ou de normalização. Nesse estudo, obteve-se com a melhor rede para classificação de figuras, uma acurácia de 90,8% para o conjunto de teste. Já para a classificação de notas, a melhor rede alcançou 85,2%. Em ambos os casos as melhores redes foram do tipo convolucional. Por outro lado, a rede MLP não teve desempenho eficaz. A correta classificação dos elementos de partituras lança as bases para a digitalização completa dessas representações musicais. Além disso, as redes neurais abordadas podem ser utilizadas em outras aplicações, como reconhecimento de padrões em imagens e sons, ou problemas de otimização.

Palavras-chave: *Deep* Learning; Rede Neural Convolucional; *Perceptron* Multicamada; Música; Partitura; Inteligência Artificial; Redes Neurais.

ABSTRACT

The importance of music for humanity is notorious, as it has influenced different aspects of societies for centuries. In addition to its cultural value, it can contribute to cognitive development and is related to a market of great financial importance. However, the diversity, volume and complexity of musical records make maintaining and manipulating sheet music a challenge. To digitize them, manual processes can be time-consuming and prone to errors. Therefore, the use of automatic sheet music recognition methods is a pertinent objective. In this context, the present work aims to explore the application of neural networks in classifying elements of musical scores. To perform this task, multilayer perceptron networks and convolutional neural networks are used. Several configurations are tested to evaluate the influence of the parameters. Several configurations were tested to assess the influence of parameters such as the number of samples per batch, number of training epochs, kernel dimensions, and number of layers, considering convolutional, dense, dropout, and normalization layers. In this study, it was achieved with the best network for figure classification, an accuracy of 90.8% for the test set. Whereas for note classification, the top-performing network reached 85.2%. In both cases, the best-performing networks were convolutional. On the other hand, the MLP network did not exhibit effective performance. The correct classification of sheet music elements lays the foundation for the complete digitization of these musical representations. Furthermore, the neural networks discussed can be used in other applications, such as pattern recognition in images and sounds, or optimization problems.

Keywords: Deep Learning; Convolutional Neural Network; Multilayer Perceptron; Music; Sheet Music; Artificial Intelligence; Neural networks.

LISTA DE ILUSTRAÇÕES

FIGURAS

QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN - C*onvolutional Neural Networks* - Rede Neural Convolucional

CSV - *Comma-separated values*

Hz - Hertz

IA - Inteligência Artificial

JSON - *JavaScript Object Notation*

IDE - *Integrated Development Environment* - Ambiente de desenvolvimento integrado

MLP - *Multilayer Perceptron* - Perceptron Multicamadas

MSE - Erro Quadrático Médio

OCR - *Optical Character Recognition* - Reconhecimento Ótico de Caracteres

OMR - *Optical Music Recognition* - Reconhecimento Ótico de Música

RGB – *Red, Green, Blue* - Vermelho, Verde, Azul.

RMSE - Erro Quadrático Médio da Raiz

RNA - Rede Neural Artificial

SGD - *Stochastic Gradient Descent* - Gradiente Descendente Estocástico

Sumário

CAPÍTULO I – INTRODUÇÃO

A primeira sessão do trabalho de conclusão de curso consiste em uma introdução sobre a proposta, juntamente com a problematização, objetivos, limitações e justificativa da relevância do trabalho.

Desde os primórdios da humanidade, a música esteve presente, não só como manifestação cultural, mas como forma de comunicação e interação, uma vez que está presente em diversos aspectos das sociedades, como questões religiosas, sociais, políticas, culturais e educativas. Assim, influenciou de forma significativa, diversas civilizações e épocas (Zotto, 2018, p. 22).

Neste sentido, a música evoluiu para estilos diferentes como *Rock and Roll*, *Pop*, *Blues, Funk Americano,* Música Popular Brasileira, músicas orientais e diversos outros. Hoje a indústria fonográfica detém um dos maiores faturamentos mundiais com uma arrecadação aproximada de US$26.2 Bilhões segundo a *International Federation of the Phonographyc Industry* (*Global Music Report 2023*, 2023, online).

Além disso, pesquisadores constataram a eficiência da música em diversas áreas, tais como desenvolvimento cognitivo e aprendizado infantil (Barreto, 2005, p. 26). E também a sua importância como metodologia didática para ensino, descrito no trabalho de (Silva, 2015, p. 27).

Para a facilitar o processo de reconhecimento, reprodução e manipulação de músicas, sistemas de escrita musical foram criados e aprimorados. Dentre eles, o sistema de partituras moderno destaca-se pela popularidade e versatilidade. Esse sistema de representação musical tem seu desenvolvimento atribuído há várias pessoas, destacando-se entre elas, Guido d'Arezzo, um representante da igreja católica do século X (Sousa, 2012, p. 50).

A partitura desempenha um papel essencial no âmbito musical ao servir como um meio universal de comunicação, preservar o patrimônio cultural e educar músicos. Além de proporcionar uma linguagem comum para músicos de diversos países, a partitura é uma ferramenta vital na transmissão de composições ao longo das gerações, garantindo a autenticidade das obras. Sua importância na educação musical é notável, pois oferece uma estrutura para o aprendizado teórico e prático, enquanto sua utilidade na interpretação musical contribui para a preservação da diversidade artística e cultural. Em última análise, a partitura é

um elo que conecta passado, presente e futuro, enriquecendo a experiência humana por meio da expressão musical.

A quantidade de partituras produzidas, e a necessidade de manipulá-las, seja para conservar, reproduzir, aprender ou editar, torna a digitalização destas representações musicais uma tarefa importante. Assim, a utilização de métodos de reconhecimento automáticos em partituras é uma metodologia pertinente, visto que pode tornar o processo mais rápido, além de eliminar possíveis falhas humanas.

Neste contexto, a inteligência artificial (IA) é um campo da ciência da computação que busca criar sistemas capazes de imitar a inteligência humana, permitindo que computadores executem diversas tarefas, como o reconhecimento de imagens. Dentre os métodos de IA, destacam-se as redes neurais artificiais (RNAs), que é um modelo computacional inspirado no funcionamento do cérebro humano, composto por camadas de unidades interconectadas, chamadas neurônios artificiais, que processam informações (Cozman; Plonski; Neri, 2021, p. 31). Essas redes neurais são um método de aprendizado de máquina, pois permitem que os sistemas reconheçam padrões complexos nos dados, aprendam com exemplos passados e ajustem suas próprias operações de forma adaptativa. Tais redes estão contribuindo para a solução de problemas como reconhecimento de padrões em imagens, processamento de linguagem natural, previsões e tomada de decisões, impulsionando avanços significativos na capacidade das máquinas de executar tarefas complexas de maneira eficiente e autônoma (Cozman; Plonski; Neri, 2021, p. 18). Portanto, Parte superior do formulário utilizando as RNAs, é possível gerar imagens, reescrevê-las, identificar padrões, e aplicá-las em diferentes situações, como por exemplo na música (Chuanzhen Li; Juanjuan Cai, 2018, p. 1).

Considerando a necessidade de reconhecimento automático de partituras, e o sucesso na aplicação de redes neurais artificias para o reconhecimento de imagens, nesse trabalho, é proposto a utilização de RNAs para o reconhecimento de elementos de partituras, identificando as figuras musicais, bem como as notas relacionadas às figuras. Para as tarefas de classificação são determinadas as acurácias dos métodos utilizados com diferentes parâmetros. E com os resultados de cada método, será feita uma análise sobre sua aplicabilidade e a influência de alguns parâmetros no projeto proposto.

No contexto supracitado, a aplicabilidade da técnica de processamento digital de imagens conhecida como Reconhecimento Ótico de Caracteres (OCR) (Manage *et al.*, 2020, p.1) pode ser extensível ao ambiente musical em forma de reconhecimento de folhas de partitura. No entanto, tal extrapolação deve considerar certas especificidades, como a

diversidade em tamanhos e formas dos elementos presentes em partituras, cujo significado muda radicalmente com a adição de pequenos detalhes ou simplesmente pela mudança de posição.

Portanto, busca-se com o auxílio de algoritmos de IA, precisamente de algumas redes neurais, identificar elementos de partituras. Isso, lança as bases para que trabalhos futuros possam digitalizar e reproduzir uma partitura de maneira precisa. E assim, substituir a necessidade humana de conhecer o que está escrito nas folhas, auxiliando leigos e aspirantes a aprenderem músicas, bem como especialistas a manipularem partituras com mais facilidade.

1.2 TEMA DO PROJETO

No presente Projeto Final de Curso foi desenvolvido um estudo relativo a redes neurais, analisando sua capacidade de interpretar figuras musicais e classificá-las de acordo com sua representação, além de também identificar as notas, considerando as posições de tais figuras no pentagrama. Para comparação de resultados, nesse trabalho foram utilizadas as Redes Neurais Convolucionais (CNNs – C*onvolutional Neural Networks*) e as Redes Neurais Perceptron Multicamadas (MLP – *Multilayer Perceptron*), demonstrando a eficiência de cada uma delas.

1.3 PROBLEMATIZAÇÃO

Explorar o avanço dos sistemas de reconhecimento e reprodução de partituras musicais emerge como um campo no qual há desafios a serem superados, especialmente ao considerar as limitações presentes nos sistemas atuais de conversão de partituras impressas para formatos digitais. Eles enfrentam dificuldades na precisão da transcrição, sobretudo em partituras complexas com variações de escrita, estilos musicais distintos, ou devido à deterioração do papel (Mannis, 2017, p.5).

Uma contribuição se concentraria em desenvolver algoritmos de reconhecimento óptico, explorando métodos de inteligência artificial, processamento digital de imagem e aprendizado de máquina com redes neurais. Tal trabalho implementaria as bases para contribuir com a preservação e edição de obras musicais, facilitando a conversão precisa e acessível de partituras impressas para formatos digitais editáveis, beneficiando músicos, editores e pesquisadores. Além disso, também poderia contribuir com a evolução musical e aprendizado aos aspirantes

da área, facilitando e contextualizando tanto alunos quanto profissionais em relação ao que é apresentado na folha de partitura.

1.4 OBJETIVOS

1.4.1 Objetivos Gerais

O objetivo geral é explorar o uso de redes neurais artificiais na classificação de figuras e notas musicais.

1.4.2 Objetivos Específicos

O objetivo geral foi alcançado seguindo os seguintes objetivos específicos:

- Estudo breve da evolução da música e a formação do seu sistema de escrita;
- Estudo das técnicas de inteligência artificial e processamento digital de dados utilizadas, com foco no reconhecimento de imagens, como as redes neurais;
- Estudo da linguagem *Python* e de suas bibliotecas necessárias para a execução do trabalho;
- Avaliação do desempenho de todo o processo, destacando a influência de parâmetros e metodologias no desempenho das redes neurais, considerando o contexto de aplicação.

1.5 LIMITAÇÕES E ESCOPO

Não é escopo deste trabalho produzir os melhores modelos de reconhecimento de partituras, visto que tal objetivo demandaria recursos computacionais, tempo de pesquisa e volume de dados que não são viáveis nas circunstâncias em que este trabalho foi desenvolvido. No entanto, destaca-se que o estudo apresentado lança as bases para a utilização de métodos mais complexos e resultados mais ambiciosos. Além disso, esclarece, de maneira prática, diversos aspectos relacionados à modelagem de redes neurais e reconhecimento de imagens.

Portanto, neste contexto, é interessante destacar que os métodos apresentados foram aplicados em imagens de elementos de partitura isolados. Isto é, não foram empregados em partituras completas, em que há vários elementos numa mesma imagem. Esses métodos poderiam, na forma como foram apresentados neste trabalho, serem utilizados para

classificação de elementos após uma segmentação em *bounding boxes*, como a existente no *dataset DeepScoreV2* (Tuggener *et al.*, 2021, p. 9189).

Outra limitação está relacionada ao conjunto de dados. Apesar de limitado em número de amostras e ter uma distribuição não homogênea de imagens por classe, permite avaliar diversos aspectos relacionados às redes neurais e proporcionar um esboço das dificuldades presentes em conjuntos de dados maiores.

1.6 JUSTIFICATIVA

O reconhecimento e reprodução de partituras é uma área relevante, pois se situa na interseção entre música e tecnologia. A necessidade de preservar, editar e compartilhar obras musicais em formatos digitais impulsiona a busca por sistemas mais eficientes e precisos de conversão de partituras impressas ou não editáveis, para meios digitais que possibilitem fácil manipulação, viabilizando, por exemplo, ações de edição ou reprodução (Calvo-Zaragoza; Jr.; Pacha, 2020, p. 1).

Os sistemas existentes enfrentam desafios na transcrição precisa de partituras complexas, limitando sua utilidade para músicos, desde o nível amador ao profissional. Além disso, o acesso a tais ferramentas de conversão é restrito, seja pela dificuldade em utilizá-las, seja pela cobrança do serviço de conversão. Um exemplo seria a ferramenta Guitar Pro, cuja assinatura após período de teste, chega ao valor de USD$69,95, cerca de R$344,96 (Arobas Music, 2024, online).

Portanto, a melhoria e a popularização desses sistemas, por meio do desenvolvimento de algoritmos de reconhecimento óptico mais avançados, aplicados às partituras, torna-se não apenas pertinente, mas também fundamental para a preservação e acessibilidade das obras musicais. Ele facilitará a edição, análise e compartilhamento desses materiais de maneira mais precisa e eficiente.

1.7 CONSIDERAÇÕES FINAIS

Nesse primeiro capítulo foi feita uma breve introdução do projeto desenvolvido, e foram destacados os seguintes pontos: principais objetivos, limitações, as justificativas que embasam a realização de tal projeto, além de quais problemas o mesmo propõe resolução. No Capítulo II, abordar-se-á o referencial teórico dessa pesquisa.

CAPÍTULO II – REFERENCIAL TEÓRICO

Nessa seção do trabalho de conclusão de curso serão apresentadas as bases referenciais que irão compor o mesmo, de maneira a detalhar o estudo realizado e introduzir cada campo de pesquisa integrado ao tema proposto.

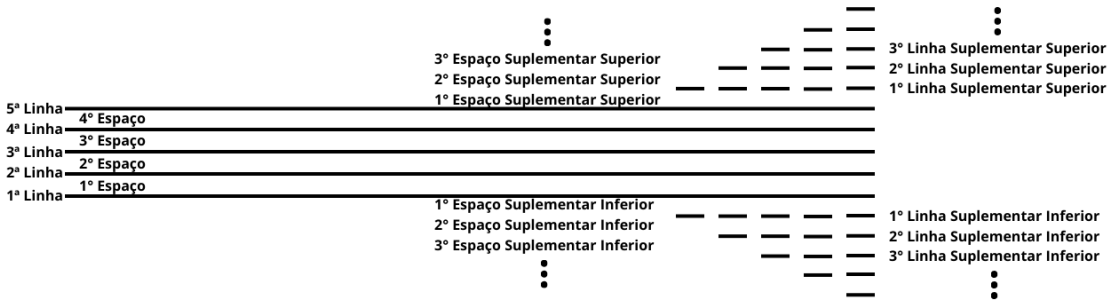
2.2 PARTITURA E SEUS ELEMENTOS

O sistema de escrita denominado partitura veio com a intenção de servir como guia e até mesmo como fórmula para que uma determinada melodia pudesse ser reproduzida em qualquer lugar do mundo de maneira original, desde que alguém consiga interpretá-la. A partitura consiste em diversos elementos, e os principais são o pentagrama, as claves, o compasso, a armadura de clave e as notas com seus acidentes e suas respectivas figuras que determinam sua duração no tempo. Na Figura 2, tem-se um exemplo de uma partitura antiga, datada em 1971, chamada "*Noche de Tampico"* escrita por Rolf Marbot para o instrumento trompete. Na partitura apresentada, observa-se vários elementos que serão abordados, como as notas, pentagrama e a clave. A partitura da figura apresenta vários sinais de desgaste que podem dificultar a identificação de notas.

Pentagrama

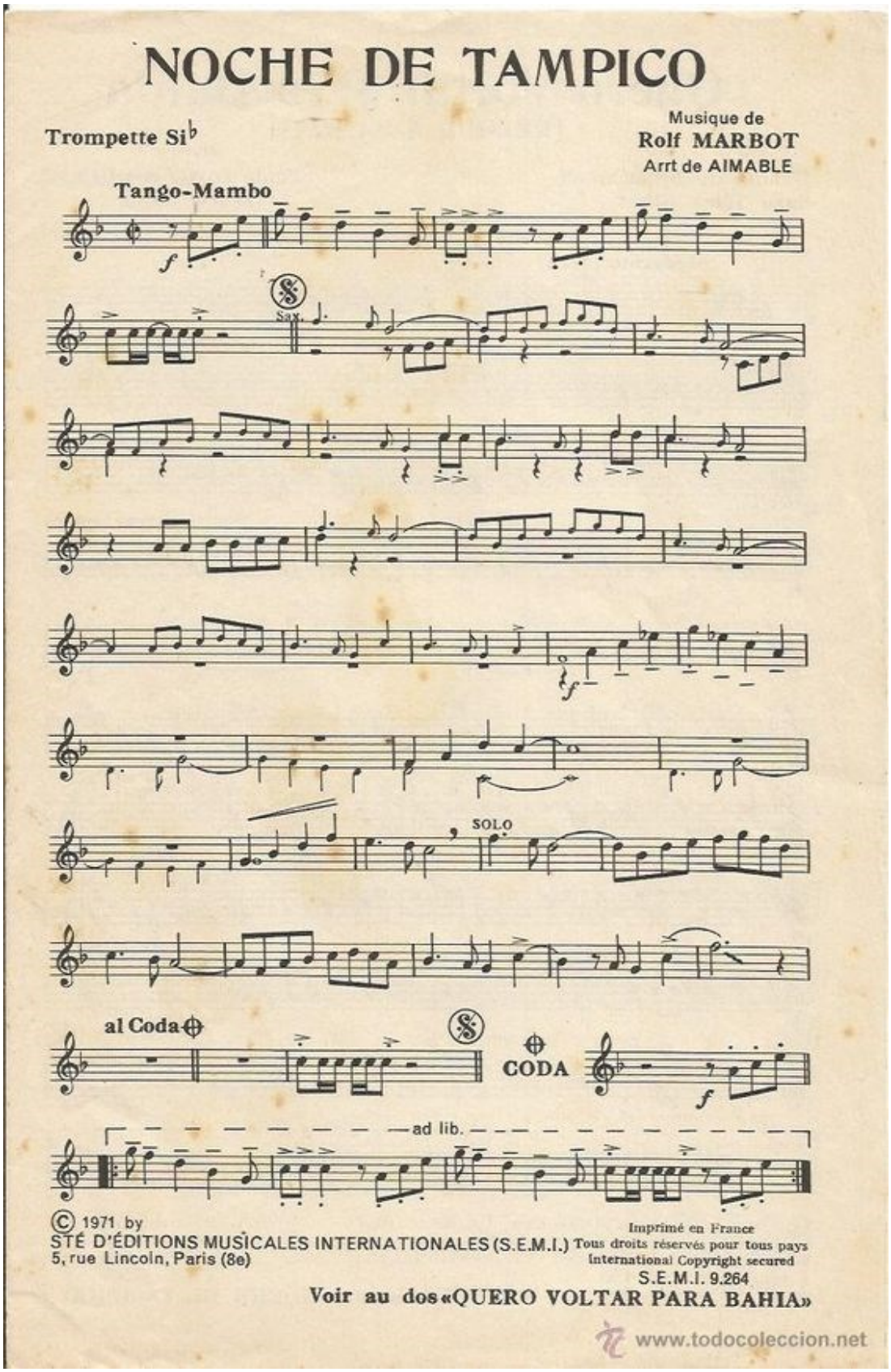
O pentagrama é o conjunto de 5 linhas horizontais, as quais servem como projeção para a alocação das figuras de notas e acidentes. Ele é subdivido em linhas e espaços como ilustrado na Figura 1, podendo conter linhas adicionais, e consequentemente espaços adicionais, que são denominadas linhas ou espaços suplementares superior ou inferior (Schmeling, 2015).

Figura 1 - Pentagrama/Pauta



Fonte: O autor.

Figura 2 - Partitura Musical Noche de Tampico.



Fonte: Marbot, 1971 (online)

Clave

É uma das figuras que define como as notas serão escritas dentro do pentagrama. Cada nota escrita pode variar de acordo com a clave que está escrita, além de tais claves serem diferentes dependendo do instrumento. Há diferentes tipos de clave, sendo as principais:

- A clave de Sol, ilustrada na Figura 3, é para a maior parte dos instrumentos com som médios e agudos como os instrumentos de corda (violão, violino e piano), e alguns instrumentos de sopro (Oboé, clarinete e trompete). A definição para a clave de Sol tem por referência a nota escrita na segunda linha, cuja representação é a própria nota Sol;
- A clave de Fá (Figura 3) que é utilizada para instrumentos com o som mais grave como o piano, contrabaixo elétrico, contrabaixo acústico e violoncelo. A definição para a clave de Fá é a quarta linha cuja nota escrita nesta linha é a própria nota Fá.
- Há ainda a clave de Dó que é utilizada para alguns instrumentos de sopro e cordas como os já citados. Tendo como adendo, vale ressaltar que a maioria dos instrumentos podem ser escritos em diferentes claves, sendo a mudança necessária conforme a música (SCHMELING, 2015).

Figura 3 – Claves

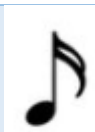
Fonte: O autor.

Compasso

Um compasso é uma unidade de medida de tempo, que divide uma peça musical em partes iguais. Cada compasso é marcado por uma barra vertical, que indica o início e o fim do compasso. A duração de um compasso é determinada pela fórmula de compasso, que é uma combinação de números que indicam o número de batidas e o tipo de nota que deve ser tocada em cada batida. O numerador indica a quantidade de batidas por compasso e o denominador indica qual figura terá o valor para corresponder a uma batida. Por exemplo, um compasso 4/4 é um compasso que tem quatro batidas por compasso, e cada batida deve ser tocada por uma nota de um quarto de nota (semínima). As figuras e durações de acordo com o compasso podem ser vistas no Quadro 1. Os compassos são usados para organizar a música e facilitar a leitura das

partituras. Eles também ajudam os músicos a manterem o ritmo da música (Schmeling, 2015, p. 13).

Quadro 1 - Figura de referência para um compasso

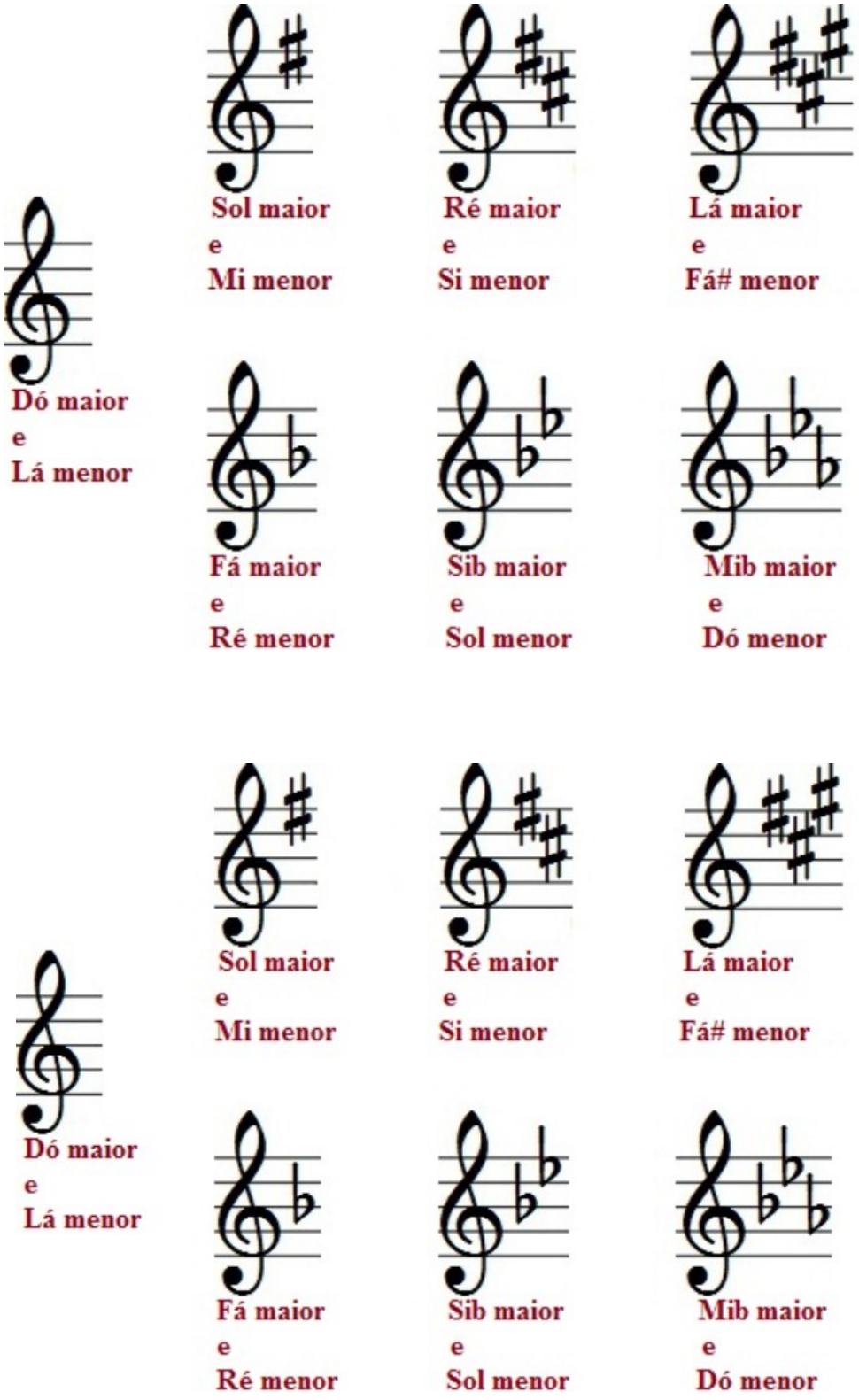
Figura da Nota	Nome	Duração
	Semibreve	1
	Mínima	2
	Semínima	4
	Colcheia	8
	Semicolcheia	16
	Fusa	32
	Semifusa	64

Fonte: O autor

Armadura de Clave

A armadura de clave tem por objetivo indicar a escala musical que será utilizada. A escala é definida pela quantidade de acidentes que há dentro da mesma, determinando as notas de dó a si (dó, ré, mi, fá, sol, lá e si), que possuem como acidentes (Schmeling, 2015, 34), como mostrado na Figura 4.

Figura 4 - Escalas definidas pela armadura de escala



Fonte: Barcellos, 2014, (online).

Notas Musicais

As notas musicais são um som que ecoam em uma determinada frequência. A frequência de uma nota é medida em hertz (Hz), e é o número de vezes que uma onda sonora completa um ciclo por segundo (Goto, 2009, p. 1). As notas musicais são representadas por símbolos, que são escritos em uma pauta musical. Além disso, são divididas em dois grupos: as notas naturais e as notas alteradas. As notas naturais são as notas que estão na escala diatônica, que é a escala básica da música ocidental. As notas alteradas são as notas que estão fora da escala diatônica, que são acompanhadas por seu respectivo acidente (sustenido ou bemol). A combinação de notas musicais cria melodias e harmonias.

A escala musical ocidental é baseada em uma progressão geométrica, com razão de frequências entre notas consecutivas fixa. Na escala temperada, a razão é a raiz duodécima de 2 (~1.0595), que define a relação de frequências entre notas adjacentes (goto, 2009, p. 1). A distância entre duas notas consecutivas é conhecida como um semitom na escala temperada. Doze semitons formam uma oitava, de tal forma que a frequência para a 12ª nota, é o dobro em relação à nota mais grave. Essa estrutura é baseada em proporções matemáticas. As notas musicais também podem ser relacionadas ao comprimento de onda do som. Notas mais graves possuem comprimentos de onda maiores, enquanto as agudas têm comprimentos de onda menores. Isso está relacionado à forma como percebemos a altura do som (Goto, 2009, p. 1).

2.3 LINGUAGEM *PYTHON*

Derivada da linguagem C, a linguagem *Python* tem como premissas simplicidade, clareza e legibilidade (Peters, 2004, online), o que contribui para a entrada de novos aspirantes à programação (Nyakundi, 2023, online; Sérvio, 2021, online). Criado em 1991 e gerenciado pela *Python Software Foundation*, é considerado "*open source"*, ou seja, código aberto, no qual qualquer pessoa pode utilizar do mesmo e aprimorá-lo conforme convém.

Dentre suas aplicações de sucesso, destaca-se a automação de tarefas, deixando-as mais simples e práticas de serem executadas, podendo funcionar a partir de simples entradas de informações ou com o apertar de botões. Contudo, a parte intuitiva do programa depende totalmente de o programador entender e implementar a necessidade daqueles que procuram seu *software* como solução.

Há linguagens que são capazes de se integrar e complementar suas funções com outras, como por exemplo, o próprio *Python* e C. Assim, firma-se a dependência de utilizar diferentes

métodos para se comunicar com máquinas, posto que cada linguagem de programação tem sua área e dispositivos de aplicação específicos.

Para o projeto, a linguagem de programação apresentada será utilizada para a implementação de redes neurais por meio de diversas bibliotecas, sendo as principais *Tensorflow* e *Keras*. A biblioteca *Keras* consiste em uma abstração de alto nível que facilita a utilização do *tensorflow* e outras bibliotecas de *machine learning* (Chollet, 2021, p. 68).

2.4 REDES NEURAIS

Uma rede neural é um modelo computacional inspirado no funcionamento do cérebro humano. Ela é composta por unidades interconectadas, chamadas neurônios artificiais, organizadas em camadas (Babini; Marranghello, 2007, p.35). Cada neurônio recebe entradas, processa essas informações e gera uma saída que pode ser transmitida para outros neurônios. Essa estrutura permite que as redes neurais aprendam padrões e relações complexas a partir dos dados, sendo amplamente utilizadas em áreas como reconhecimento de padrões, processamento de linguagem natural, visão computacional (Babini; Marranghello, 2007. 25), entre outras aplicações. Essa capacidade de aprendizado e generalização faz das redes neurais um dos pilares da inteligência artificial moderna.

2.4.1 Neurônios Artificiais

Um neurônio artificial é a unidade fundamental de processamento em uma rede neural artificial. Inspirado na estrutura dos neurônios biológicos do cérebro humano, um neurônio artificial recebe múltiplas entradas, cada uma multiplicada por um peso específico (Babini; Marranghello, 2007, p. 25). Estes pesos são ajustáveis e representam a importância relativa de cada entrada para o neurônio. No *perceptron*, um tipo de neurônio artificial largamente utilizado, a soma ponderada das entradas, incluindo um viés (*bias*), é aplicada a uma função de ativação, que determina a saída do neurônio. Essa saída pode ser transmitida para outros neurônios na rede neural (Babini; Marranghello, 2007, p. 38).

A seguir, mostra-se a função soma (Eq. 1), em que x são as entradas e w os pesos. Esta soma é posteriormente aplicada à uma função de ativação (Huang *et al.*, 2020, p. 4).

$$soma = \sum_{i=1}^{n} x_i * w_i \quad \text{(Eq. 1)}$$

Neste contexto:

- Peso positivo: sinapse excitadora;
- Peso negativo: sinapse inibidora;
- Pesos amplificam ou reduzem o sinal de entrada;
- O conhecimento da rede neural são os pesos.

Para o treinamento de uma rede neural, deve-se encontrar os pesos para os atributos. Para a classificação de parafusos, por exemplo, teríamos:

- Atributos: comprimento e diâmetro (x_1 e x_2) são as entradas;
- Pesos: são encontrados de tal forma que o melhor conjunto de peso é aquele que permite classificar corretamente os parafusas nas classes A ou B, com base nas entradas;
- Saída: classes (A ou B).

2.4.2 Tipos de Redes Neurais

As redes neurais, assim como outros algoritmos de aprendizado de máquina, podem ser divididas em três tipos diferentes, conforme o seu método de aprendizagem.

Na aprendizagem supervisionada, a rede neural é treinada com um conjunto de dados rotulados. Isso significa que cada entrada de dados é associada a uma saída desejada (Géron, 2017, p. 28). Durante o treinamento, a rede compara sua saída com a saída desejada e ajusta seus parâmetros para minimizar a diferença entre esses valores. Um exemplo comum é o treinamento de uma rede neural para reconhecer dígitos escritos à mão. Cada imagem de um dígito é associada a um rótulo que indica qual dígito (0 a 9) ela representa. A rede é treinada com várias dessas imagens e seus rótulos correspondentes, aprendendo a associar corretamente as características das imagens aos rótulos.

Nas redes neurais não-supervisionadas, a rede é treinada em dados não rotulados, e o objetivo é encontrar estruturas, padrões ou relações nos dados sem a orientação de rótulos externos (Géron, 2017, p. 28). Um exemplo comum é o uso de redes neurais para realizar tarefas de agrupamento (*clustering*). Por exemplo, ao alimentar a rede com dados de clientes de uma empresa (sem informação prévia sobre categorias), ela pode aprender a agrupar os clientes com características semelhantes em grupos distintos. Isso pode revelar segmentos de mercado ou comportamentos comuns entre os clientes;

No aprendizado por reforço, a rede neural aprende através de interações com um ambiente, tomando ações e recebendo recompensas ou penalidades em resposta a essas ações (Géron, 2017, p. 28). O objetivo é maximizar a recompensa cumulativa ao longo do tempo. Por exemplo, em um jogo, a rede aprende a jogar melhor experimentando diferentes ações e recebendo *feedback* (recompensa ou penalidade) com base no desempenho. Com o tempo, ela aprende a tomar as ações que levam a melhores resultados de acordo com a recompensa esperada.

2.4.3 Redes Multicamadas

Redes neurais multicamadas são redes que consistem em três ou mais camadas de neurônios interconectados: uma camada de entrada, uma ou mais camadas ocultas *(hidden layers)* e uma camada de saída. Cada camada é composta por neurônios artificiais que se conectam aos neurônios da camada seguinte por meio de conexões ponderadas, nas quais cada neurônio é conectado com todos os neurônios da próxima camada. Este método é denominado de conexão densa (Chollet, 2021, p. 2).

Camada de Entrada

Recebe os dados que serão processados pela rede. Cada neurônio nesta camada representa um atributo ou uma característica dos dados de entrada.

Camadas Ocultas

São camadas intermediárias entre a camada de entrada e a camada de saída. Cada neurônio nessas camadas processa informações das camadas anteriores, realizando cálculos e transformações nos dados. A presença de múltiplas camadas ocultas permite que a rede capture e aprenda representações cada vez mais complexas dos dados.

No entanto o aumento no número de neurônios e camadas aumenta o tempo de treinamento e a necessidade de mais recursos computacionais. Assim, a definição desses parâmetros não é uma escolha trivial, mas há, na literatura, algumas regras que podem nortear tal escolha. Por exemplo, pode-se definir a quantidade de neurônios de cada camada seguindo a fórmula da Eq. 2 (Shibata; Ikeda, 2009, p. 2):

$$N_N = \frac{\sqrt{N_i N_o}}{2} \qquad \text{(Eq. 2)}$$

em que N_i é o número de entradas, ou neurônios de entrada, e N_o é o número de saídas. Quanto mais complexo o problema, mais camadas ocultas podem ser necessárias.

Camada de Saída

Produz os resultados da rede após processar as informações das camadas ocultas. Dependendo da tarefa, pode haver um único neurônio de saída ou múltiplos neurônios que representam diferentes classes, valores numéricos, ou outras saídas desejadas.

Geralmente, problemas complexos exigem mais de um neurônio na saída, melhorando a eficiência da classificação das respostas. O conceito de *deep learning* parte do princípio de haver mais de uma camada oculta até chegar na saída. Contudo, o problema do *deep learning* é o "problema do gradiente desaparecendo" (*vanishing gradiente problem*). Tal erro acontece em redes neurais profundas quando os gradientes dos pesos das camadas mais profundas tornam-se extremamente pequenos durante o treinamento, dificultando a atualização adequada dos pesos e prejudicando o aprendizado nessas camadas. Estratégias como a inicialização cuidadosa dos pesos, o uso de funções de ativação apropriadas e arquiteturas alternativas ajudam a mitigar esse problema, permitindo um treinamento mais eficaz de redes neurais profundas.

2.4.4 Ajuste de pesos, funções de perda e algoritmos de otimização

Funções de Custo (*Loss Function*)

A rede neural ajusta os pesos para aprender sobre os padrões encontrados nos dados. Para isso é necessário calcular os erros. De maneira simples, o erro pode ser calculado pela diferença entre a resposta correta e a estimada pelo modelo (Chollet, 2021, p.254). Para adequar e agrupar os erros de várias amostras, funções de perda (*loss function*) são estabelecidas como métricas de erro. Então algoritmos de otimização são utilizados para atualizar os pesos com base nas funções de custo. Neste trabalho, os termos custo e perda serão utilizados para designar tais funções.

Dentre as várias medidas de erro, que podem ser utilizadas para avaliar os modelos e também como função de custo, podemos destacar o *mean square error* (MSE – erro quadrático médio) e o *root mean square error* (RMSE – raiz do erro quadrático médio).

O MSE, que está descrito na terceira equação (Eq. 3), ou Erro Quadrático Médio, é uma métrica comum para avaliar a performance de modelos de regressão. Ele mede a média dos quadrados das diferenças entre os valores preditos pelo modelo e os valores reais dos dados

(Babini; Marranghello, 2007, p. 35). Quanto menor o valor do MSE, melhor o modelo está em prever os dados de maneira precisa. É uma medida que penaliza fortemente erros maiores, fornecendo uma visão clara da qualidade geral das previsões do modelo.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2 \quad \text{(Eq. 3)}$$

Já o RMSE (Eq. 4) é uma variação do MSE que fornece a raiz quadrada da média dos quadrados das diferenças entre os valores preditos e os valores reais dos dados. Essa métrica é útil pois retorna uma medida de erro na mesma unidade dos dados originais, facilitando a interpretação. Quanto menor o valor do RMSE, melhor o desempenho do modelo em prever os dados de maneira precisa, similar ao MSE, mas expresso de forma mais intuitiva por estar na mesma escala dos dados originais.

$$RMSE = \sqrt{MSE} \quad \text{(Eq. 4)}$$

Em problemas de classificação, outras funções de custo são utilizadas, como a c*ategorical cross-entropy* (Mao; Mohri; Zhong, 2023, p. 1).

Gradiente (Máximos e Mínimos) e Algoritmos de Otimização

O gradiente refere-se à taxa de variação de uma função em relação às suas variáveis. Em uma rede neural, o objetivo é minimizar uma função de perda, que mede o quão distantes as previsões da rede estão dos valores reais. Para minimizar essa função, o gradiente descendente (descida do gradiente) é um método largamente utilizado. Esse método consiste em um procedimento iterativo que ajusta os pesos da rede na direção oposta ao gradiente da função de perda. Isso leva a encontrar os mínimos locais ou globais da função de perda.

A atualização dos pesos segue uma determinada ordem, como pode ser visto na Figura 5. Desta forma, pode-se ocorrer considerando uma ou várias amostras. Neste sentido temos: o Gradiente Descendente Estocástico (SGD) que atualiza os pesos da rede para cada exemplo de treinamento individual; o Gradiente Descendente em *Batch* que atualiza os pesos após calcular a função de perda considerando todos as amostras; e o Gradiente Descendente em *Mini-Batches* que divide os dados de treinamento em conjuntos menores (*mini-batches*) para calcular e atualizar os pesos. O SGD evita mínimos locais, mas é instável. Já a atualização por *Batch* pode resultar em mínimos locais, além de convergência lenta. Por outro lado, a utilização de *Mini-Batches* combina eficiência e estabilidade na convergência do treinamento. Este último

é amplamente usado na prática, por oferecer um equilíbrio entre eficiência computacional e estabilidade na otimização de redes neurais.

Dentre os algoritmos de otimização, podemos destacar além do gradiente descendente, o Adam (Kingma; Ba, 2014, p. 1). Este consiste em uma evolução do primeiro, para facilitar a adaptação a diferentes funções de erro, e o ajuste de parâmetros, como a taxa de aprendizado.

Back propagation

É o algoritmo fundamental usado para calcular o gradiente da função de perda em relação aos pesos da rede. Ele opera de forma iterativa, propagando o erro da camada de saída para as camadas anteriores, usando a regra da cadeia para calcular como cada peso contribui para o erro. Esse processo permite ajustar os pesos em todas as camadas da rede, com base no erro cometido na previsão.

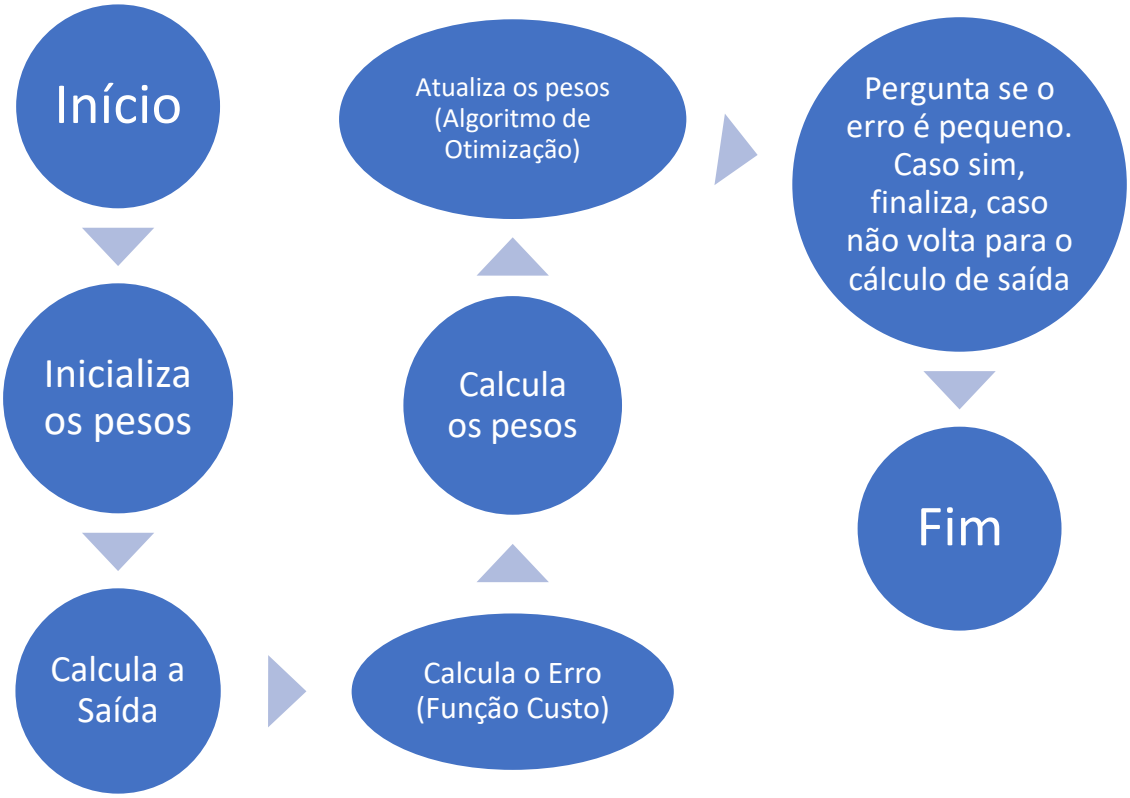
Taxa de Aprendizado (Learning Rate)

É um hiper parâmetro importante no treinamento de redes neurais. Ele determina a magnitude dos ajustes nos pesos da rede a cada passo do gradiente descendente. Um *learning rate* muito pequeno pode resultar em convergência lenta, enquanto um muito grande pode fazer com que a otimização oscile ou até mesmo seja divergente do resultado esperado.

Momentum

O momentum é uma técnica usada para acelerar o gradiente descendente, especialmente em áreas planas ou onde há curvas suaves. Ele introduz um termo que acumula o histórico das atualizações dos pesos anteriores e adiciona um fator de inércia na direção do movimento, o que pode ajudar a superar mínimos locais e a alcançar mais rapidamente a região de mínimo global. Se o momento for alto, aumenta a velocidade de convergência, mas diminui a eficiência, se for baixo, pode evitar os mínimos globais, mas demora mais para convergir.

Figura 5 - Algoritmo para cálculo e ajuste dos pesos



Fonte: O autor

2.5 DEEP LEARNING

O *Deep Learning* é um ramo da inteligência artificial que se baseia em algoritmos de aprendizado de máquina para modelar e entender dados complexos. Ele é caracterizado pelo uso de redes neurais profundas, compostas por múltiplas camadas.

2.5.1 *Perceptron* Multicamadas *vs* Rede Neural Convolucional

O *Perceptron* Multicamadas (MLP) é uma arquitetura de rede neural composta por uma camada de entrada, uma ou mais camadas ocultas (que processam os dados de maneira não linear) e uma camada de saída. Cada neurônio em uma camada está conectado a todos os neurônios na camada seguinte, permitindo que a rede aprenda relações complexas nos dados. O treinamento do MLP é geralmente feito utilizando o algoritmo de *backpropagation*

(Aggarwal, 2018, p. 21), onde os pesos da rede são ajustados iterativamente para minimizar uma função de perda, utilizando *gradient descent*.

Já as Redes Neurais Convolucionais (CNN) são uma categoria especializada de redes neurais projetadas para processar dados com estruturas de grade, como imagens, tendo como entrada os pixels da mesma. Tais redes tiveram grande sucesso em reconhecimento de padrões em imagens (Sultana; Sufian; Dutta, 2018, p. 1) e têm sua origem nos trabalhos de autores como Kunihiko Fukushima (Fukushima, 1980, p. 1), Alex Waibel (Waibel *et al.*, 1989, p. 1), e Yann LeCun (Lecun et al., 1989, p. 1).

Elas consistem em camadas convolucionais, de *pooling* e, frequentemente, camadas totalmente conectadas. As camadas convolucionais aplicam filtros a pequenas áreas das imagens para extrair características como bordas, texturas e padrões. As camadas de *pooling* reduzem a dimensionalidade dos mapas de características resultantes. As CNNs são amplamente utilizadas em tarefas de visão computacional, como reconhecimento de objetos, segmentação de imagens e detecção de padrões (Aggarwal, 2018, p. 326). Além disso, são analisados os parâmetros, detalhes e características de uma determinada figura e com eles, o sistema pode classificar e denominar ao que a respectiva imagem se refere. Por exemplo a classificação de um animal ser um gato ou um cachorro.

Em resumo, O MLP é uma arquitetura básica de rede neural, composta por camadas totalmente conectadas, onde cada neurônio em uma camada está conectado a todos os neurônios da camada seguinte. Essa estrutura torna o MLP versátil e aplicável a uma variedade de tarefas, como classificação, regressão e aprendizado de padrões em dados não estruturados. No entanto, ao lidar com dados como imagens, o MLP enfrenta desafios devido à perda de informações espaciais, visto que ele não considera a estrutura espacial dos dados.

Já a CNN é especialmente projetada para processar dados com estruturas de grade, como imagens. Ela consiste em camadas convolucionais, de *pooling* e as camadas densas no final. As camadas convolucionais aplicam filtros em regiões locais dos dados, capturando características espaciais e preservando a relação entre os pixels. Essa capacidade de capturar padrões locais e preservar a estrutura espacial torna as CNNs altamente eficazes em tarefas de visão computacional, como reconhecimento de objetos, segmentação de imagens e detecção de padrões.

Enquanto o MLP é mais genérico e pode ser aplicado em diferentes domínios, as CNNs são altamente especializadas para processar imagens devido à sua capacidade de extrair e preservar características locais. Embora ambos sejam poderosos métodos de *Deep Learning*, as

CNNs se destacam em tarefas de visão computacional, enquanto o MLP é mais versátil e aplicável a uma gama mais ampla de problemas de aprendizado de máquina.

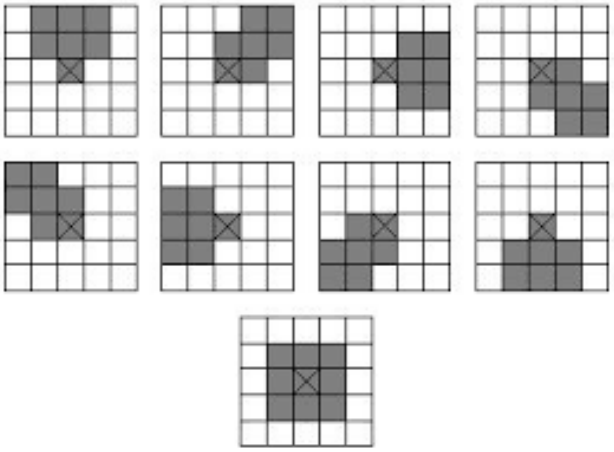
Uma rede neural convolucional não utiliza todos os pixels como entradas de cada *kernel* simultaneamente. Na verdade, os *kernels*, equivalentes de neurônios para as camadas convolucionais, possuem dimensões menores, e são deslocados por toda a imagem. Isto permite extrair as principais as características em qualquer lugar da imagem com poucos pesos, de tal forma a economizar processamento computacional. Uma arquitetura de rede neural convolucional típica contém os seguintes elementos: (Aggarwal, 2018, p. 318):

- Etapa 1 – Operador de Convolução;
- Etapa 2 – *Pooling;*
- Etapa 3 – *Flattening;*
- *Etapa 4* – Rede neural densa.
-

Operador de convolução

A convolução é um conceito matemático importante que tem aplicações em diversas áreas, incluindo processamento de sinais, processamento de imagens, redes neurais, entre outros. Em termos simples, a convolução é uma operação matemática que combina duas funções para produzir uma terceira função, que representa como uma delas "influencia" a outra ao longo de um intervalo de valores. Tendo como exemplo imagens, um processo de convolução adicionará cada elemento da imagem para seus vizinhos, ponderado por uma janela (*kernel)* (Figura 6). Este método é possível visto que uma imagem é interpretada computacionalmente como uma matriz, desta forma um determinado *kernel* pode alterar a imagem original alterando seus valores matriciais originais.

Figura 6 - Método de convolução por kernel



Fonte: Baldo; Rezende, 2004 (online).

Pooling

A etapa de *pooling* é crucial nas Redes Neurais Convolucionais (CNNs). Geralmente é inserida entre as camadas convolucionais. Seu objetivo principal é reduzir a dimensionalidade dos mapas de características obtidos nas camadas convolucionais, preservando as informações essenciais (Aggarwal, 2018, p. 326).

Existem duas técnicas principais de *pooling*: *Max Pooling* e *Average Pooling.* O *Max Pooling* seleciona o valor máximo em cada região de um mapa de características, reduzindo a dimensionalidade enquanto preserva características salientes. Por outro lado, o *Average Pooling* calcula a média dos valores na região, útil para suavizar informações e reduzir ruídos nos dados. O *Max Pooling* tem sido mais utilizado que o *Average Pooling* (Aggarwal, 2018, p. 120).

Essa camada de *pooling* tem alguns objetivos principais. Um deles é reduzir a quantidade de parâmetros e cálculos nas camadas seguintes, o que diminui o tempo de processamento e evita o *overfitting*. Além disso, ela ajuda a preservar informações importantes e as hierarquias de características aprendidas nas camadas convolucionais ao reduzir a resolução espacial dos mapas de características.

Flattening

A etapa de *Flattening* é frequentemente utilizada após as camadas convolucionais ou de *pooling* em Redes Neurais Convolucionais (CNNs). Essa etapa transforma os mapas de

características (*feature maps*) multidimensionais em um vetor unidimensional, preparando os dados para a passagem pela rede neural densa (*fully connected layer*) subsequente (Aggarwal, 2018).

Durante a convolução e *pooling*, os mapas de características mantêm a informação espacial das imagens, sendo representados em 3D (altura x largura x canais). O *Flattening* converte tais mapas em um único vetor, onde cada elemento representa uma característica ou um valor específico desses mapas. Esse vetor unidimensional resultante é então passado através de camadas densas (*fully connected layers*), onde cada neurônio está conectado a todos os neurônios da camada anterior, permitindo a aprendizagem de padrões mais complexos e a geração de saídas finais da rede.

O *Flattening* é bastante importante para conectar as camadas convolucionais ou de *pooling* às camadas densas finais da rede neural, permitindo que informações relevantes dos mapas de características sejam processadas e utilizadas para a tomada de decisão ou classificação final.

Ambos os métodos, *Perceptron* Multicamadas e Redes Neurais Convolucionais, são exemplos poderosos de arquiteturas de redes neurais profundas, cada uma especializada em diferentes tipos de dados e tarefas. O *Deep Learning*, por meio dessas e de outras arquiteturas, tem revolucionado áreas como visão computacional, processamento de linguagem natural, medicina, entre outros, permitindo o processamento e a compreensão de dados complexos de forma eficaz e automática (Santos, 2019).

2.5.2 Validação cruzada

A validação cruzada é uma metodologia importante para avaliação de modelos, sendo o método de *k-fold cross-validation* comumente utilizado. Nesse método, os dados de treinamento são divididos em k subconjuntos, e o modelo é treinado k vezes, cada vez utilizando k-1 *folds* como conjunto de treinamento e 1 *fold* como conjunto de validação. O desempenho do modelo é avaliado calculando a média das métricas obtidas nas k iterações. Essa técnica é vantajosa para estimar o desempenho real do modelo, selecionar hiper parâmetros e garantir sua capacidade de generalização para novos conjuntos de dados (Yadav; Shukla, 2016).

De maneira simples, em uma base grande de registros é normal utilizar 90% dos dados como fontes para treinamento e 10% para validação deles. Já em uma base de dados menor, essa porcentagem pode variar de acordo com o necessário. Contudo, ainda é possível que dados que serão utilizados para a validação sejam mais eficientes como treinamento. A validação

cruzada serve para resolver este impasse. Neste método, os dados são divididos em K_{partes}, desta maneira x partes são utilizadas para teste enquanto y partes fica para testes, após isso, as partes para treinamento e testes se alternam. No exemplo da Figura 7, temos $K = 4$. Assim, os dados são divididos em 4 partes e o processo é feito 4 vezes. O resultado é a média aritmética simples dos resultados (Yadav; Shukla, 2016) conforme a Equação 5:

$$Resultado = \frac{x+y+z+w}{K}. \quad \text{(Eq. 5)}$$

Figura 7 - Exemplo de Validação Cruzada

Fonte: O autor

2.5.3 *Underfitting* e *Overfitting*

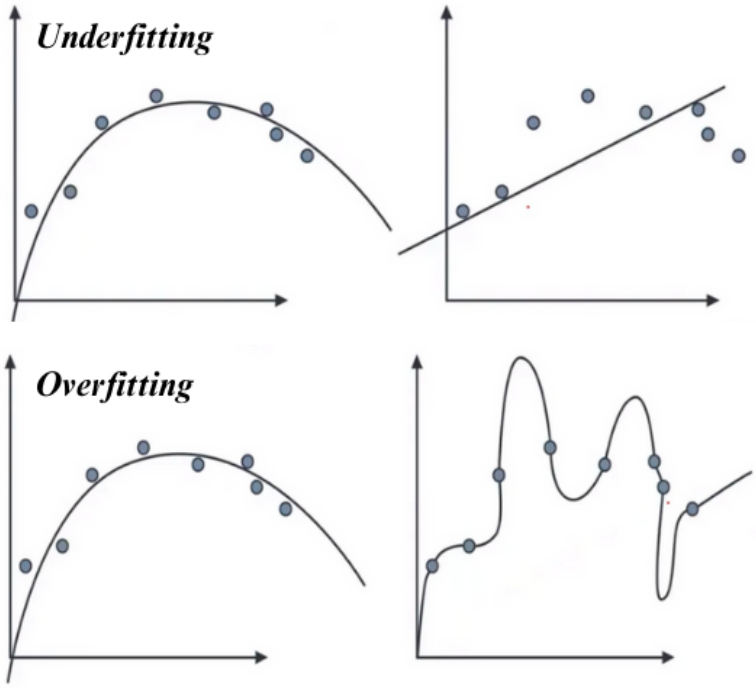
No aprendizado de máquina utilizado redes neurais há dois problemas que são bastante comuns. O *underfitting* é uma subestimação do problema. Neste cenário, utilizando um algoritmo ou parâmetros mais simples do que o necessário para tratar um problema complexo, obtem-se resultados ruins na base de treinamento (Babini; Marranghello, 2007). Um exemplo é a utilização de uma reta para representar dados em uma parábola (Figura 8).

Em contrapartida o *overfitting* é caracterizado por se utilizar um algoritmo complexo e diversos recursos para tratar um caso simples, em que não é necessária uma lógica tão elaborada, resultando em boas análises na base de treinamento, porém, resultados ruins na base de teste. Isto acontece porque o modelo se adapta especificamente a base de treinamento (Chollet, 2021). Desta maneira, caso haja algum dado que esteja um pouco fora do padrão, o

algoritmo não será capaz de generalizar com precisão, causando erros na variação de novas instâncias. Um exemplo é uma função complexa, como um polinômio de grau elevado, para representar dados de uma parábola (Figura 8).

Uma técnica eficaz para amenizar o *overfitting* é a *dropout* (Géron, 2017), cuja característica é anular alguns valores de entrada aleatoriamente durante o treino. Em outras palavras, a técnica consiste em atribuir um valor nulo em uma determinada porcentagem das entradas. Para esta porcentagem, recomenda-se valores entre 20% e 30%, visto que a anulação de grande parte das entradas pode levar ao problema de *underfitting*.

Figura 8 - Exemplo de Underfitting e Overfitting



Fonte: Adaptado de Wingate, 2020 (online).

2.6 OPTICAL MUSIC RECOGNITION (OMR)

Optical Music Recognition (OMR) é uma vertente no campo da visão computacional e processamento de imagem que se dedica à automação da conversão de partituras musicais, sejam impressas ou manuscritas, em notação musical digital (REBELO *et al.*, 2012). O principal propósito do OMR é extrair informações musicais de forma precisa e legível a partir de imagens de partituras, possibilitando a manipulação, análise ou reprodução dessas informações em formato digital.

O processo de OMR inclui várias etapas. Inicia-se com o pré-processamento da imagem, que envolve técnicas como binarização, remoção de ruído, realce de contraste e segmentação para separar os diferentes elementos musicais. Em seguida, há a identificação e segmentação de símbolos musicais individuais, como notas, pausas, claves e acidentes musicais (Chuanzhen Li; Juanjuan Cai, 2018).

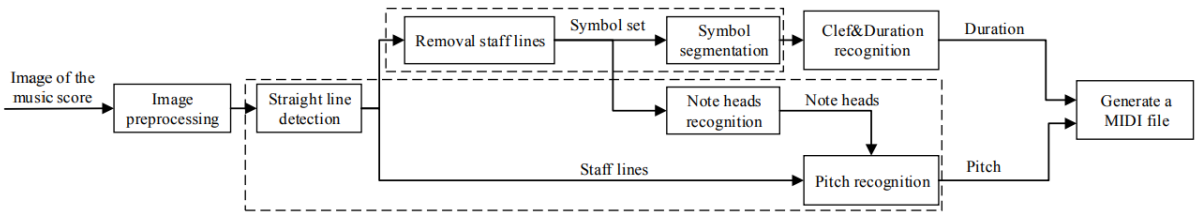
O reconhecimento óptico de caracteres (OCR) é aplicado para converter esses símbolos musicais segmentados em notação musical digital, utilizando técnicas de correspondência de padrões e identificação específica de símbolos musicais. Posteriormente, ocorre a transformação dos caracteres reconhecidos em uma representação musical mais abstrata, que pode incluir informações sobre altura das notas, duração, dinâmica, articulação e outros elementos musicais (jimit k. Shah, 2019).

O OMR é aplicado em situações diversas como digitalização de partituras antigas, criação de bancos de dados musicais digitais, auxílio na transcrição de músicas manuscritas e integração com softwares de notação musical. Apesar dos desafios relacionados à diversidade de estilos musicais, complexidade das partituras e variações na qualidade da digitalização, os avanços em técnicas de aprendizado de máquina e processamento de imagem têm contribuído para aprimorar a eficiência e precisão desse sistema.

2.6.1 Pesquisas correlatas

Há diferentes métodos e propostas para a implementação do OMR em um sistema neural. Um método que se provou eficiente foi o descrito por (Chuanzhen Li; Juanjuan Cai, 2018) (Figura 9):

Figura 9 - Proposta de sistema OMR por Chuanzhen Li e Juanjuan Cai



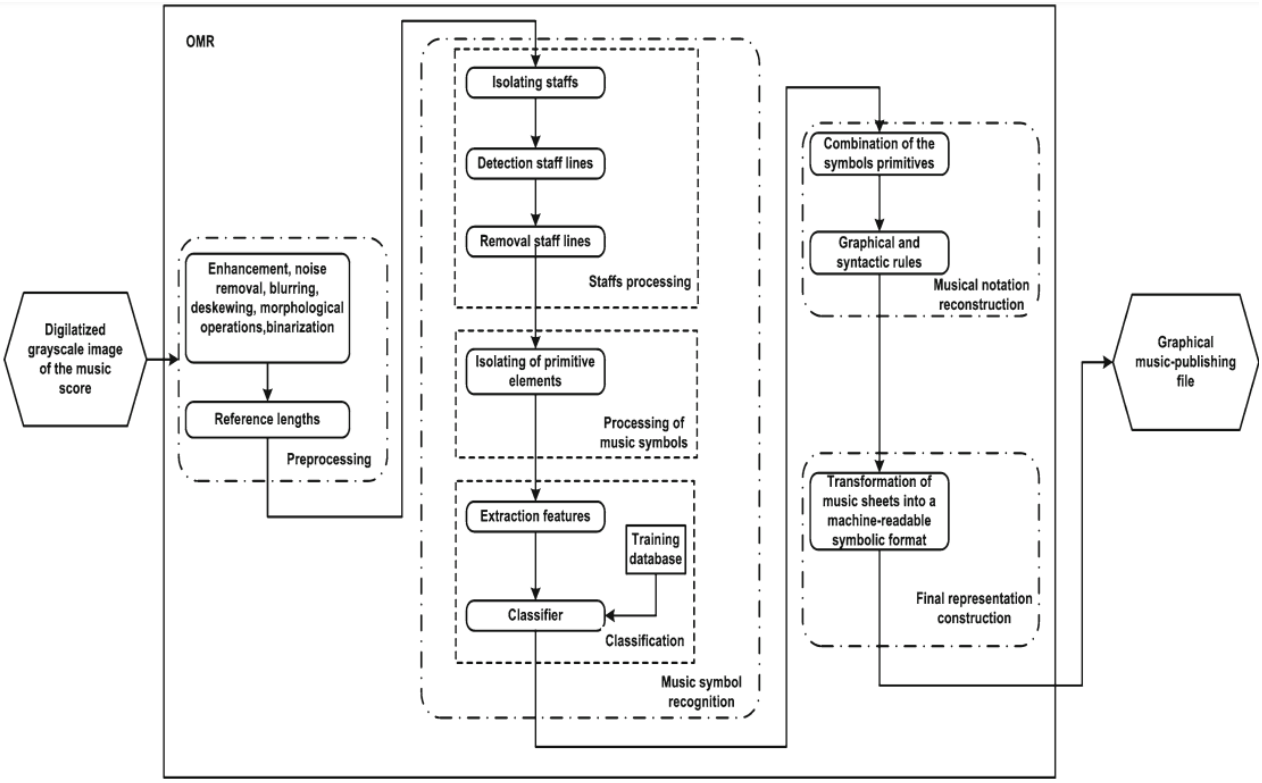
Fonte: Chuanzhen Li; Juanjuan Cai, 2018 (p. 2)

Neste modelo há várias etapas nas quais contam inicialmente com o processamento da imagem, detecção de linhas retas (as pautas), remoção das linhas, reconhecimento das figuras musicais, das notas musicais, reconhecimento da clave e duração dos tempos, conversão da altura das notas (tonalidade de acordo com a clave) e término com a geração do arquivo MIDI para que *softwares* possam interpretar o que foi lido na partitura.

Já para (Rebelo *et al.*, 2012) o sistema conta com as mesmas características, contudo, havendo mais detalhes como o tratamento da imagem com a remoção de ruídos, binarização, tratamento de efeitos de suavização, a combinação dos símbolos para formar estruturas mais complexas e a transformação dos dados para uma representação específica de leitura de máquina, geralmente em binário, como é possível visualizar na Figura 10.

Sendo estes os métodos mais tradicionais, há também variações mínimas que podem contribuir para a melhora ou piora do aprendizado de máquina, como a detecção de tom da partitura antes da identificação da pauta, figuras e notas, alteração na ordem de detecção dos elementos e os métodos de construção para linguagem de máquinas (Bellini; Bruno; Nesi, 2022).

Figura 10 - Arquitetura típica para o modelo de OMR



Fonte: Rebelo *et al.*, 2012 (p. 2)

Para o projeto de conclusão de curso apresentado, realizou-se a construção de dois tipos de redes neurais: rede neural de perceptrons multicamadas e rede neural convolucional. Em ambos os casos, utilizou-se as redes para classificar elementos da partitura já segmentados.

2.7 CONSIDERAÇÕES FINAIS

Abordou-se neste capítulo o referencial teórico que compõe o escopo do trabalho de conclusão de curso apresentado, tendo por direcionamento os tópicos de partitura e sistema de escrita musical, linguagem de programação *Python*, redes neurais, *deep learning* e *optical music recognition* (OMR). Na próxima sessão abordar-se-á os materiais e métodos necessários para a realização do projeto, comparação e validação dos resultados.

CAPÍTULO III – MATERIAL E MÉTODOS

Neste capítulo será apresentado como o projeto foi realizado, a metodologia utilizada, além dos materiais como softwares e bibliotecas.

O Projeto foi desenvolvimento seguindo as seguintes etapas:

I. Realização de pesquisas teóricas em livros didáticos e nos trabalhos desenvolvidos por especialistas nas áreas de escrita musical, redes neurais, classificação de imagens e OMR, que estão relacionadas aos objetivos deste trabalho;
II. Realização de estudos da linguagem de programação *Python* e suas bibliotecas para a sintetização da rede neural que dará base para a obtenção de resultados que irão compor o escopo do trabalho;
III. Estabelecimento do *dataset* de imagens a ser utilizado e implementar computacionalmente o tema proposto utilizando os métodos de rede neural convolucional e perceptrons multicamadas para a comparação e análise dos resultados.

Além disso, os principais recursos utilizados para implementar o software proposto foram: o *Integrated Development Environment* (IDE) *Visual Studio Code*, que é gratuito e *open source*; e bibliotecas da linguagem *Python* como *tensorflow* e *keras*.

3.1 Pesquisas teóricas

Inicialmente, foram pesquisados os principais artigos científicos que de áreas como escrita musical, *deep learning*, inteligência artificial e reconhecimento musical óptico (OMR), principalmente trabalhos acadêmicos que englobam todos estes temas. Tendo por base principal e mais relevante o OMR, decidiu-se utilizar os seguintes artigos: *"Optical Music Recognition: State of The Art And Major Challenges"* de Elona Shatri e Gyorgy Fazekas (Shatri; Fazekas, 2020), *"Optical Music Recognition: State-Of-The-Art And Open Issues"* de Ana Rebelo et. al. (Rebelo *et al.*, 2012), *"OMR Reader Info Scanner"* de Sparsh Agarwal, Malempati Varun e S Prabakeran (Agarwal; Varun; Prabakeran, 2023), *"From Optical Music Recognition to Handwritten Music Recognition: A baseline"* escrito por Arnau Baró, Pau Riba, Jorge Calvo-Zaragoza e Alicia Fornés (Baró *et al.*, 2019). No contexto de redes neurais, as principais fontes

foram o livro “Introdução às Redes Neurais Artificiais” escrito por Maurizio Babini e Norian Marranghello e disponibilizado pela Universidade Estadual Paulista (UNESP) (Babini; Marranghello, 2007), além dos livros escritos por Francois Chollet, *“Deep Learning with Python, Second Edition”* (Chollet, 2021), “*Hands-On Machine Learning with Scikit-Learn & TensorFlow*” de Aurélien Géron (Géron, 2017) e “*Neural Networks and Deep Learning*” do escritor Charu C. Aggarwal (Aggarwal, 2018). Além disso, também foram realizados cursos que contribuíram com conhecimento prático, e consequentemente, auxiliaram na construção e execução das redes neurais.

3.2 Linguagem e bibliotecas de programação

A linguagem de programação *Python* foi escolhida para implementar as redes neurais considerando a larga utilização da mesma no contexto de ciência de dados e *machine learning*. As principais bibliotecas utilizadas foram o “Keras”, que tem por finalidade auxiliar na construção das arquiteturas de redes neurais, com funções e comandos de alto nível para definir um tipo de rede neural, como serão as camadas, quantos neurônios terão, tipo de entrada, otimizadores, métricas, função de perda, função de ativação dentre outros parâmetros, além de possibilitar o treinamento e validação de resultados. Vale ressaltar que uma das bibliotecas mais comuns para a utilização em redes neurais é denominada “Tensorflow”. Contudo a biblioteca Keras é uma derivação da Tensorflow e visa ser mais simples e direta. Alguns módulos da biblioteca “Scikit-learn” também foram utilizados. Para manipular números e vetores, foi-se utilizada tanto a biblioteca “NumPy” quanto a biblioteca “Pandas” que são comuns para a tarefa de trabalho numérico.

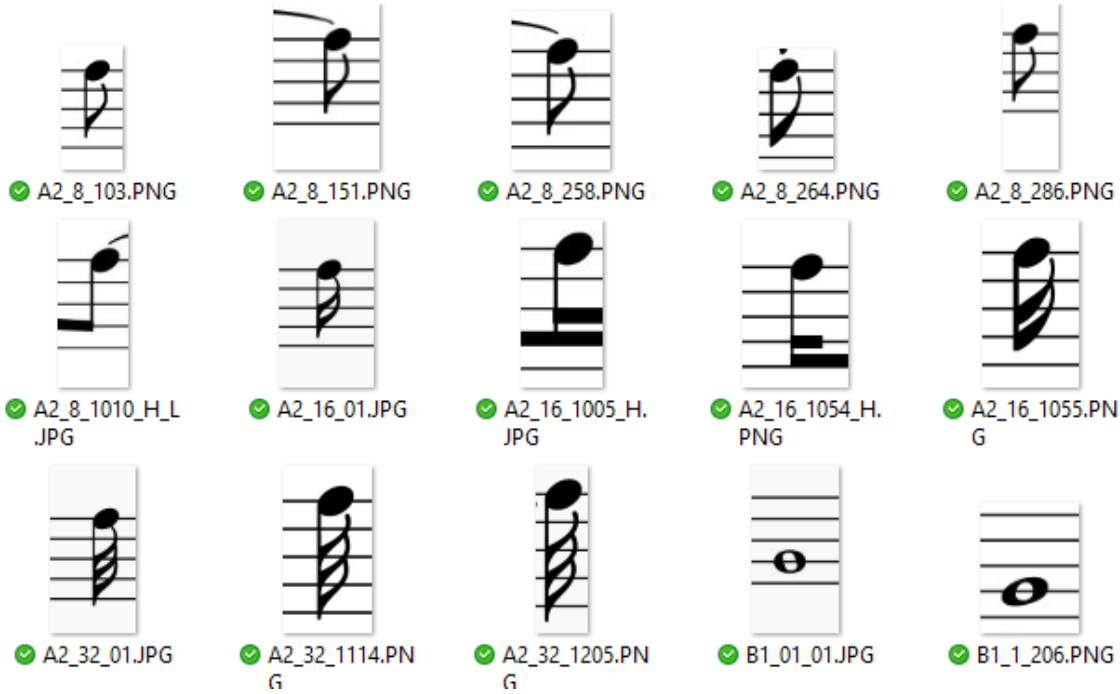
3.3 Conjunto de dados

Os dados utilizados para treino e teste do modelo consistem em um *dataset* próprio de imagens construído pelo autor. Este conjunto de dados consiste em 542 imagens utilizadas para treino e validação cruzada, e outras 141 imagens utilizadas somente para teste, totalizando 683 imagens de elementos de partitura. Algumas amostras dos conjuntos de treino e teste são mostradas na Figura 11 e Figura 12, respectivamente. Estas imagens foram extraídas de diferentes fontes e partituras.

Como mostrado na Figura 12, o *dataset* de teste possui imagens relativamente diferentes do conjunto de treino, como imagens de baixa resolução, embaçadas, com fundos coloridos,

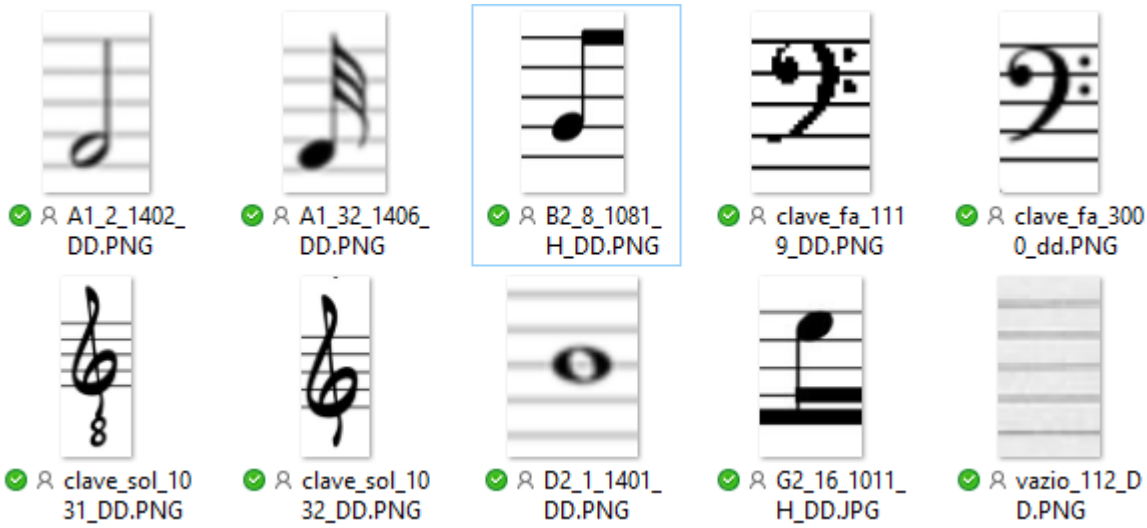
claves distorcidas etc. Isto foi feito de modo proposital, buscando torná-lo mais desafiador para os modelos, e permitir verificar aqueles com melhor capacidade de generalização.

Figura 11 – Algumas amostras do conjunto de treino e validação.



Fonte: O autor

Figura 12 – Algumas amostras do conjunto de teste.



Fonte: O autor

Para contornar o problema da quantidade de imagens limitadas, utilizou-se técnicas de *data augmentation*. Estas técnicas consistem em aplicar pequenas alterações nas imagens durante o processo de treino, como deslocar e aumentar ou diminuir (*zoom*), que não provocam

mudança de significado da imagem. Tais técnicas devem ser aplicadas considerando o contexto dos dados. Um leão, por exemplo, continua sendo um leão em qualquer ponto de uma imagem, mesmo que ela tenha sido espelhada. No entanto, no contexto de partituras, um espelhamento vertical mudaria completamente o sentido da imagem, alterando por exemplo, a nota indicada.

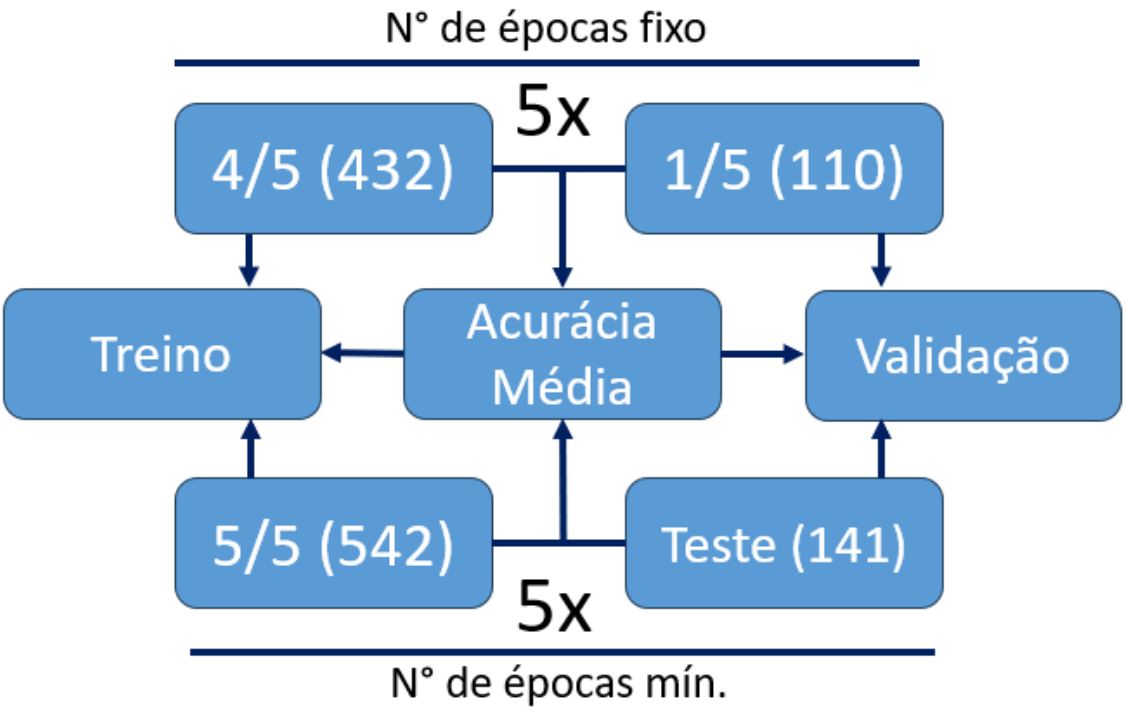
3.4 Implementação e metodologia de teste dos modelos

Os modelos utilizados neste trabalho, MLPs e CNNs, foram aplicados em dois problemas de classificação distintos: classificação de figuras e classificação de notas. O primeiro problema consiste em classificar as imagens em 11 categorias: 'bemol', 'sustenido', 'clave_de_fa', 'clave_de_sol', 'semibreve', 'minima', 'seminima', 'colcheia', 'semicolcheia', 'fusa' e 'vazio'.

O segundo problema consiste em classificar as imagens de acordo com a altura das cabeças das notas, o que determina o quão grave ou agudo deve ser a nota apresentada na figura. As notas foram representadas como se todas as imagens estivessem na clave de fá, o que pode ser facilmente convertido para outras claves, alterando-se a nota de referência. Neste problema foram utilizadas notas da segunda linha complementar inferior até a segunda linha complementar superior, representadas por 17 classes: 'C1_', 'C2_', 'C3_', 'D1_', 'D2_', 'D3_', 'E1_', 'E2_', 'E3_', 'F1_', 'F2_', 'G1_', 'G2_', 'A1_', 'A2_', 'B1_', 'B2_'. Os nomes das imagens foram colocados de tal forma que representassem as classes que cada imagem pertence, permitindo que *scripts* pudessem identificar o rótulo correto nas etapas de aprendizado supervisionado. Na Figura 11 é possível ver vários nomes com as classes das notas explícitas. As classes de figuras foram identificadas por números que indicam o denominador da fração relacionada à figura, apresentados no Quadro 1.

Para validar os modelos utilizou-se uma validação cruzada estratificada com k=5. O termo “estratificada” significa que a escolha dos elementos para cada *fold* busca manter a proporção de cada classe semelhante ao conjunto original. O número de folds foi estabelecido visando garantir que cada fold pudesse ter um número razoável de elementos de cada classe, além de não penalizar o tempo de execução, uma vez que quanto maior o número de folds, maior o número de ciclos de treino para cada configuração de modelos testada. A Figura 13 mostra um esquema que representa a metodologia de treino e validação implementada.

Figura 13 - Diagrama da metodologia implementada.



Fonte: O autor

Nesta metodologia as etapas consistem em:

1. **Divisão dos *folds*:** determinou-se um $k = 5$, para a divisão dos *folds*. Este valor foi determinado a partir de considerações como o tamanho do banco de imagens e também da quantidade de classes totais para a classificação de resultados da rede neural;
2. **Treinamento com divisão de *folds:*** nesta etapa, fez-se o treinamento utilizando 4/5 (432 imagens) do *dataset*, e 1/5 (110 imagens) do banco de dados para a validação do treinamento. Obteve-se então a acurácia para cada iteração de *folds*, tanto para o conjunto de treino quanto para o de validação, e por fim calculou-se a acurácia média para ambos;
3. **Estimação dos resultados:** realizou-se a estimação das classes para os dados do conjunto de teste, com 141 imagens. É válido ressaltar que a estimação das classes nesta etapa foi feita com o conjunto de pesos da última iteração, ou seja, utilizando o modelo treinado com os *folds* da quinta permutação do método *k-folds cross-validation*;
4. **Treinamento utilizando todos os folds:** para este treinamento, foi utilizado o conjunto total de 542 imagens para o treinamento, e o conjunto de teste, com 141

imagens, para a validação. Este segundo conjunto não foi utilizado em nenhum momento para treino. A utilização de todo o conjunto de *folds* como treino visa permitir a rede atingir a melhor performance possível no conjunto de teste. Por fim, foi extraída dos resultados a acurácia média para posterior avaliação.

Para cada configuração de modelo, após obter a média de acurácia dos conjuntos de treino e validação entre os *folds*, utilizou-se todos os *folds*, isto é, todo o conjunto de treino e validação, para treinar a rede e testá-la no conjunto de testes. A utilização de todo o conjunto como treino para posterior validação no conjunto de teste, visa permitir a rede atingir a melhor performance possível. Além disso, é importante mencionar que o número de épocas de treinamento ao utilizar o conjunto de testes foi determinado como a média das épocas com menor função perda para os conjuntos de validação dos *folds*. Isto busca diminuir a possibilidade de *overfitting* devido a um grande número de épocas de treino.

Para viabilizar o teste de várias configurações de redes neurais, criou-se uma estrutura de parâmetros armazenada formato semelhante ao *javascript object notation* (JSON), que permite alterar diversos parâmetros das redes, sem ter que programá-las diretamente. A Figura 14 mostra a estrutura de parâmetros para uma CNN.

Figura 14 – Estrutura para representação e teste de parâmetros de redes CNN.

```
ann_params = {
    'ann_type': 'cnn', # cnn ou mlp
    'description': 'cnn ', # cnn ou mlp
    'epochs': 80,
    'n_conv_layers': 2,
    'n_filters': [64, 32],
    'kernel_size': [[3,13],[13,3]],
    'max_pool_type': 'after_every_conv', # None, 'after_every_conv', 'after_every_2_conv', 'after_first_2'
    'max_pool_size': [[1,2], [2,1]],
    'dropout_conv_type': '', # None, 'after_every_conv', 'after_every_2_conv'
    'dropout_conv_probabities': [],
    'n_hidden_layers': 2,
    'n_neurons_hidden_layers':[128, 64], #
    'dropout_hidden_probabities':[0.1, 0.1], # mesmo número de elementos de n_neurons_hidden_layers
    'norm_layers': False,
    'kernel_regularizer': True
}
```

Fonte: O autor

Na Figura 15 encontra-se um trecho do código que permite criar diferentes redes CNN por meio dos parâmetros indicados. Para as MLPs, um método semelhante foi implementado.

Além disso, para cada configuração foram salvas diversas informações, como gráficos da função de perda e da acurácia em função das épocas do treino, pesos dos modelos, bem como valores numéricos de métricas de acurácia, função de perda e tempo de execução em arquivos

comma-separated values (CSV - valores separados por vírgulas), facilitando a comparação dos modelos por meio de tabelas.

Figura 15 – Trecho de código utilizado para criação dinâmica de redes CNN.

```
classificador = Sequential()
for i in range(n_conv_layers):
    # print('i')
    # print(n_conv_layers)
    if i>0: kernel_regularizer = None
    classificador.add(Conv2D(n_filters[i], kernel_size[i], input_shape = input_shape,
                             activation='relu', kernel_regularizer=kernel_regularizer) )

    if norm_layers:
        classificador.add(BatchNormalization())

    if( max_pool_type == 'after_every_conv' or
        max_pool_type == 'after_every_2_conv' and i%2==1 or
        max_pool_type == 'after_first_2' and i<2):
        j = i if max_pool_type == 'after_every_conv' else (i//2)
        classificador.add(MaxPooling2D(pool_size=max_pool_size[j]))

    if( dropout_conv_type == 'after_every_conv' or
        dropout_conv_type == 'after_every_2_conv' and i%2==1):
        j = i if dropout_conv_type == 'after_every_conv' else (i//2)
        classificador.add(Dropout(dropout_conv_probabities[j]))

classificador.add(Flatten())

for i in range(n_hidden_layers):
    classificador.add(Dense(units=n_neurons_hidden_layers[i], activation='relu'))
    classificador.add(Dropout(dropout_hidden_probabities[i]))

classificador.add(Dense(units = NUMBER_OF_CLASSES, activation = 'softmax'))
```

Fonte: O autor

CAPÍTULO IV – RESULTADOS

Neste capítulo serão apresentados e analisados os resultados obtidos após os procedimentos de treinamento e mudança de parâmetros, discutindo a eficácia dos métodos, a influência dos parâmetros e outros aspectos importantes.

4.1 Classificação de Figuras

Para a classificação de elementos em figuras foram testadas mais de 80 configurações de parâmetros. A Tabela 1 apresenta os resultados para algumas dessas configurações. As melhores redes CNN e MLP para a classificação em figuras estão presentes nessa tabela. É interessante destacar que cada métrica pode revelar características importantes das redes.

Os termos utilizados na tabela estão definidos abaixo.

- **Tipo:** tipo da rede neural que foi utilizado (MLP ou CNN);
- **ID:** identificação da rede. Muda de acordo com os parâmetros utilizado;
- **Acurácia média dos *folds* de treino:** porcentagem média de amostras dos *folds* de treinamento classificada corretamente utilizando o método de divisão de *folds* (4/5 (treino))
- **Acurácia média dos *folds* de validação:** porcentagem média de amostras do *fold* de validação classificada corretamente utilizando o método de divisão de *folds* (1/5 (validação))
- **Acurácia do conjunto de teste com os pesos de treino do último *fold*:** porcentagem correta do conjunto de teste utilizando o *dataset* específico para treino;
- **Acurácia do conjunto de teste com o treino completo:** porcentagem correta do conjunto de teste utilizando o método de treino com todos os *folds*;
- **Tempo de execução total [s]:** tempo de execução total para todas as etapas da de treino da rede (aprendizado, validação e teste);
- **Épocas de treino para divisão de *folds*: n**úmero de iterações para o treino utilizando o método da divisão de *folds*;
- **Épocas de treino para o conjunto completo:** número Iterações totais para o treino utilizando o método com todos os *folds.*

Tabela 1– Métricas para algumas configurações de modelos na classificação por figuras

Tipo	ID	Acurácia média Folds de treino	Acurácia média Folds de validação	Acurácia do conjunto de teste com os pesos de treino do último fold	Acurácia do conjunto de teste com treino completo	Tempo de Execução total [s]	Épocas treino para divisão de folds	Épocas treino para o completo
CNN	A	96,9%	90,2%	79,4%	90,8%	615	100	87
CNN	B	97,5%	87,5%	83,7%	90,8%	1088	100	89
CNN	C	98,9%	88,2%	88,7%	87,9%	2029	200	156
CNN	D	95,6%	87,1%	84,4%	87,9%	1728	80	70
CNN	G	98,8%	86,5%	78,0%	81,6%	1225	200	157
MLP	E	72,2%	52,0%	50,4%	61,0%	557	100	84

Fonte: O autor.

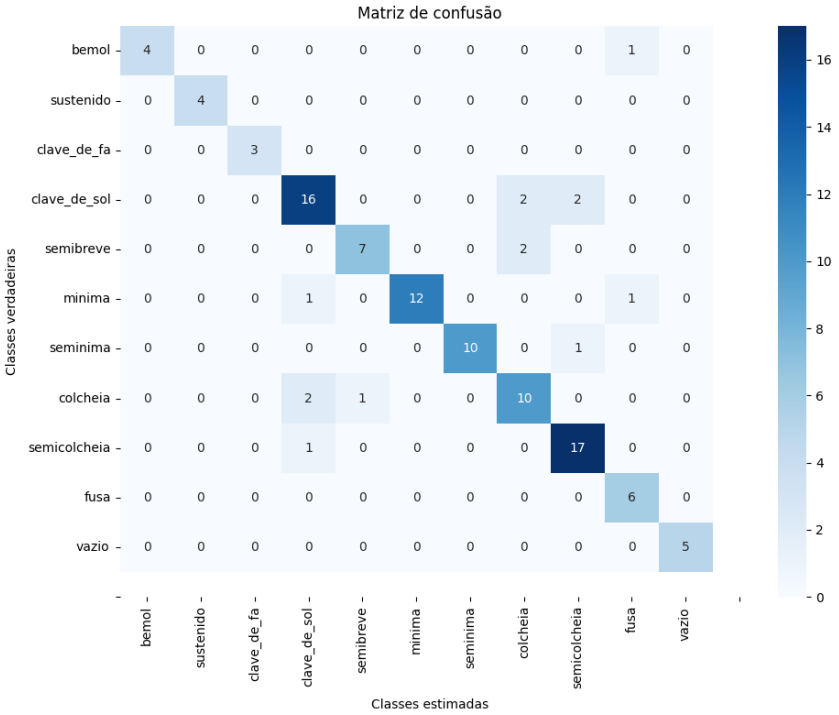
A acurácia obtida ao utilizar o conjunto de teste com treino completo, bem como a acurácia média para os *folds* de validação na etapa do *k-fold*, quarta e sexta colunas da Tabela 1 respectivamente, auxiliam na avaliação da capacidade de generalização do modelo. Afinal, como citado na seção II, se um modelo tem elevada acurácia num conjunto de treino, mas baixa acurácia em outros conjuntos, há um indicativo de *overfitting*.

A CNN G, por exemplo, apresentou quase 99% de acurácia média, nos folds de treino, mas apenas 86,5% para os *folds* de validação, e 81,6% para o conjunto de teste, considerando o treino realizado com todos os *folds*. Esse modelo foi treinado por 200 épocas nos treinos dos *folds*, e 157 para o treino com todo os *folds*. O número de épocas de treinamento elevadas é um dos motivos que podem levar ao *overfitting*. Além disso, o modelo obteve apenas 78% de acurácia no conjunto de teste com os pesos obtidos no treino com o último fold. Isso revela que, neste caso, a estratégia de diminuir o número de épocas para 157, a média das épocas de perda mínima dos treinos com os *folds*, contribuiu para uma aumentar a acurácia no treino com todos os *folds*, o que provavelmente ocorreu pela diminuição de overfitting.

Tal efeito também foi observado para a CNN A, que obteve a maior acurácia para o conjunto de teste no treino com todos os folds, uma vez que passou de 79,4%, para 90,8%, treinando 87 épocas ao invés de 100. É válido mencionar que parte dessa melhoria também pode estar relacionado ao fato de que no treino com todos os *folds*, o modelo tem mais dados para aprender os padrões.

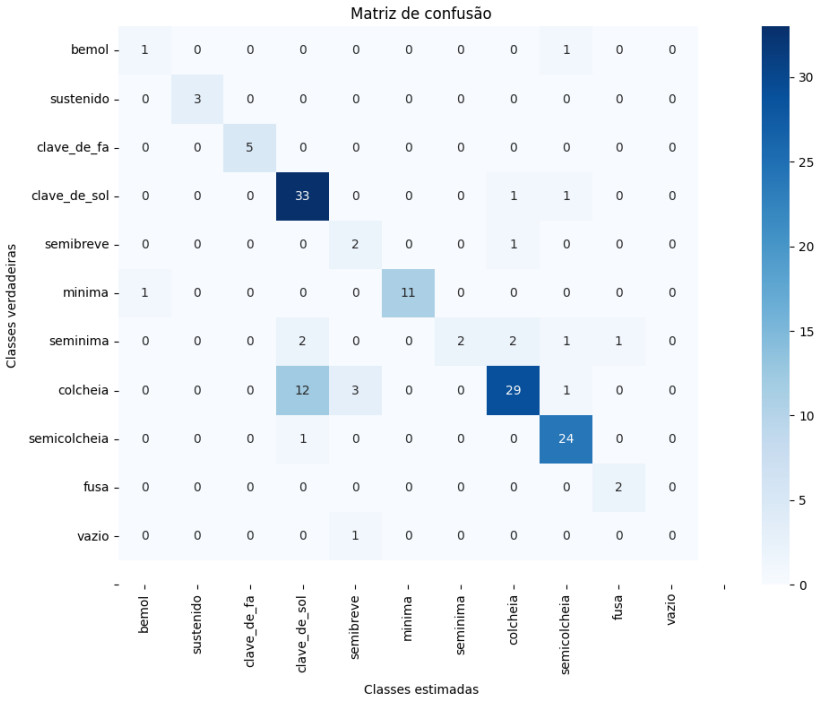
A Figuras 16, 17 e 18 mostram as matrizes de confusão para a rede CNN A, considerando respectivamente: a predição para o último *fold*; a predição para o conjunto de teste, com os pesos do treino do último *fold*; e também a predição para o conjunto de testes após o treino com todos os *folds*.

Figura 16 – Matriz de confusão da predição para o último *fold*.



Fonte: O autor

Figura 17 – Matriz de confusão da predição para o conjunto de teste, com os pesos do treino do último *fold*.



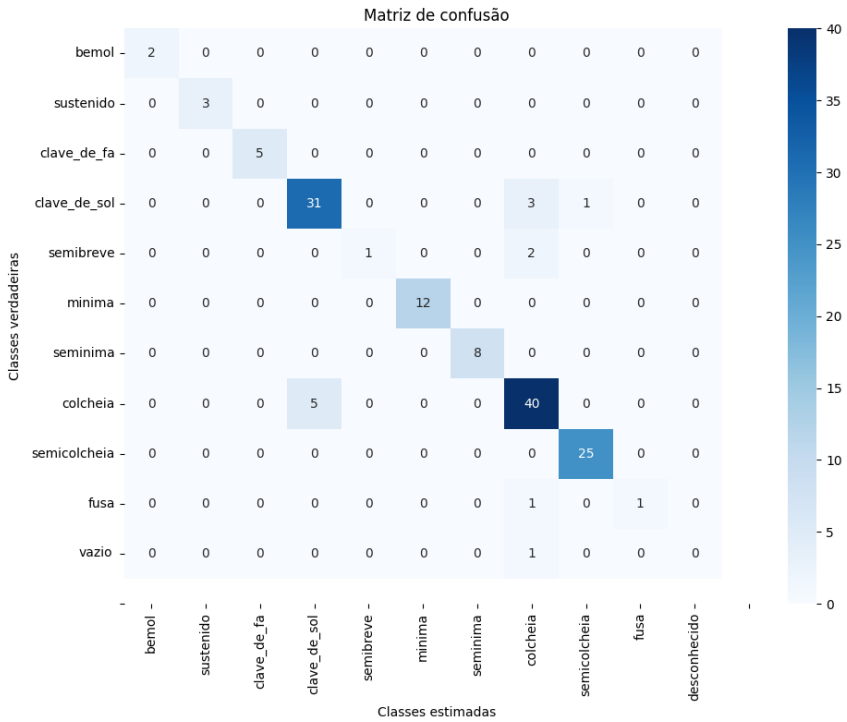
Fonte: O autor

É possível perceber que o modelo erra mais, em números absolutos, a classe das colcheias, em todos os casos. No entanto, para o conjunto de testes isso é esperado, uma vez que essa é a classe mais frequente, com 45 amostras e um erro percentual de 12,5%, considerando o treino com todos os folds no treinamento, o que pode ser visto na Figura 18. Neste exemplo, o erro percentual se refere ao número de elementos que são da classe colcheia, mas foram classificados em outras classes, o que seria equivalente à taxa de falsos negativos em uma classificação binária.

Além disso, observa-se uma melhora considerável na predição do conjunto de teste utilizando todos os *folds* no treino (Figura 18). Um exemplo dessa melhoria seria a taxa de falsos negativos da classe semínima, que foi de 75% para 0% em relação a predição feita com os pesos do treino do último *fold* (Figura 17).

É válido destacar que para a classificação por figuras, o conjunto de teste é menos balanceado que o conjunto utilizado na etapa da validação cruzada. Este desbalanceamento não afeta o treino das redes, uma vez que o conjunto de testes não é utilizado para treino. No conjunto dividido em *folds*, há algumas classes, como bemol, sustenido e clave de fá, que possuem menos amostras, mas tais classes tiveram baixas taxas de erro, o que indica que o modelo conseguiu aprender padrões, mesmo das classes com menor frequência.

Figura 18 – Matriz de confusão da predição para o conjunto de teste com todos os *folds* para treino.



Fonte: O autor

Ao analisar o tempo de execução total dos treinos na Tabela 1, observa-se que CNN A possui um tempo de execução bem menor que as outras CNNs. No entanto, CNN A e B são muito parecidas: possuem 6, 16 e 120 kernels nas três camadas de convolução, bem como kernels com dimensões iguais de [7, 7], [7, 7] e [5, 5].

A principal diferença das duas redes é que CNN A possui apenas uma camada densa com 84 neurônios após as camadas de convolução e *pooling*, enquanto CNN B possui duas camadas densas de 84 e 40 neurônios. Isso indica outro aspecto: as camadas densas, por terem pesos únicos para cada entrada em cada neurônio, podem ser computacionalmente mais "caras" que as camadas *convolucionais*, que compartilham os mesmos pesos para diferentes partes da imagem. Os principais parâmetros da rede CNN A são mostrados na Figura 19.

Figura 19 – Parâmetros para a rede CNN A.

```
'ann_type': 'cnn', # cnn ou mlp
'description': 'cnn ',
'epochs': 100,
'n_conv_layers': 3,
'n_filters': [6, 16, 120],
'kernel_size': [[7,7], [7,7], [5,5]],
'max_pool_type': 'after_first_2',
'max_pool_size': [[2,2], [2,2]],
'dropout_conv_type': '',
'dropout_conv_probabities': [],
'n_hidden_layers': 1,
'n_neurons_hidden_layers':[84], #
'dropout_hidden_probabities':[0.2],
'norm_layers': False,
'kernel_regularizer': True
```

Fonte: O autor

A diversas configurações de redes MLP foram superadas com folga por diversas CNNs. Assim, verifica-se que as redes CNN são realmente mais adaptadas para a classificação de imagens, o que está alinhado com a teoria.

Em outro cenário de testes, manteve-se uma configuração de rede neural e variou-se o número de amostras por *batch*. A Tabela 2 sintetiza as informações deste teste. Nela é possível perceber que menores tamanhos de *batch* resultaram em uma acurácia maior. O resultado deve ser analisado com cautela, considerando que é esperado que a convergência de treinos com

batch maior deve ser mais lenta. No entanto, mesmo com um número de épocas maior, configurações com *batches* de 64 e 128 não obtiveram melhorias em relação às métricas de acurácia.

Tabela 2 - Métricas para treinos com diferentes tamanhos de batch na classificação por figuras

Tamanho do *batch*	ID	Acurácia média Folds de treino	Acurácia média Folds de validação	Acurácia do conjunto de teste com os pesos de treino do último fold	Acurácia do conjunto de teste com treino completo	Tempo de Execução total [s]	Épocas treino para divisão de folds	Épocas treino para o completo
8	H	93,7%	88,2%	80,1%	84,4%	455	50	44
16	I	91,0%	85,6%	81,6%	84,4%	415	50	43
32	J	88,0%	83,2%	73,8%	78,0%	399	50	43
64	K	85,1%	76,8%	79,4%	58,2%	389	50	43
128	L	73,8%	71,0%	61,7%	58,2%	369	50	47

Fonte: O autor

4.2 Classificação de Notas

Para a classificação de elementos em notas foram testadas mais de 50 configurações de parâmetros. A Tabela 3 apresenta os resultados para algumas dessas configurações. As melhores redes CNN e MLP para a classificação em notas estão presentes nessa tabela. Mais uma vez, mas redes CNN superaram as redes MLP com facilidade.

Tabela 3 - Métricas para algumas configurações de modelos na classificação por notas

Tipo	ID	Acurácia média Folds de treino	Acurácia média Folds de validação	Acurácia do conjunto de teste com os pesos de treino do último fold	Acurácia do conjunto de teste com treino completo	Tempo de Execução total [s]	Épocas treino para divisão de folds	Épocas treino para o completo
CNN	M	97,1%	86,2%	75,8%	85,2%	1255	80	66
CNN	N	95,2%	85,6%	82,8%	80,5%	538	80	60
CNN	O	99,4%	87,2%	82,8%	77,3%	1605	200	118
CNN	A	98,8%	87,2%	80,5%	75,8%	653	100	68
CNN	P	97,5%	82,7%	71,1%	72,7%	285	50	38
MLP	Q	70,1%	55,5%	39,1%	44,5%	321	100	80

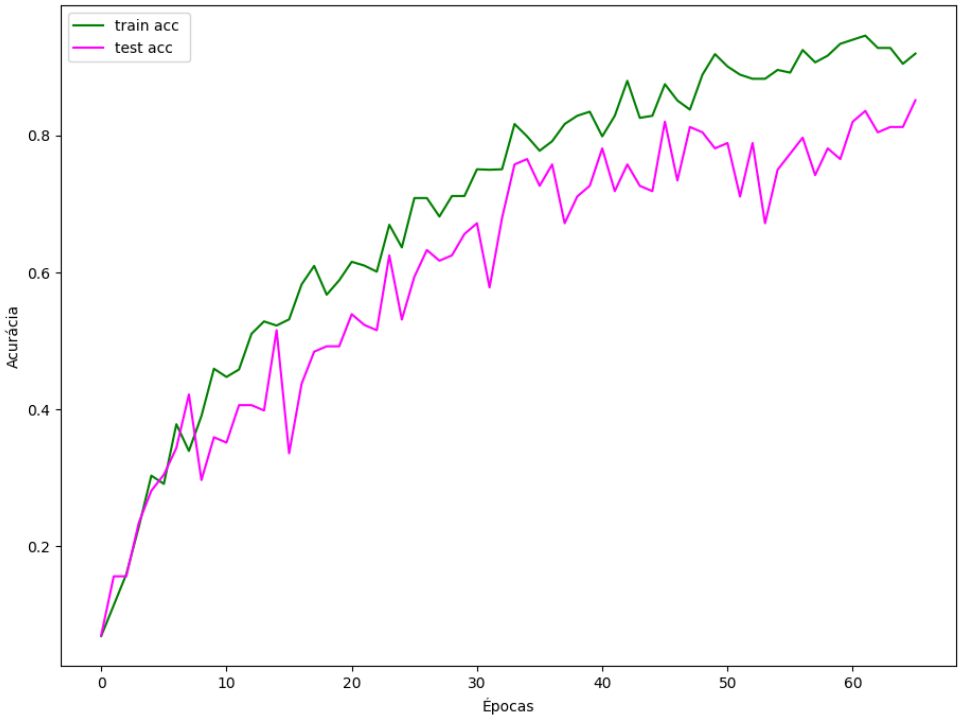
Fonte: O autor.

Mais uma vez, temos um exemplo que sugere a ocorrência de *overfitting*: a CNN O apresentou uma acurácia média de 99,4%, o que representa a maior acurácia média para os conjuntos de treino da tabela. No entanto, sua métrica de acurácia no conjunto de teste foi de

apenas 77,3%. Além disso, o número de épocas de treino da rede foi de 200 nos treinos da etapa de *k-fold* e 118 para o treino com todos os *folds*.

Conforme explicado na Capítulo III, ao treinar as redes com todos *folds*¸ o número de épocas foi definido com base na média da época com menores perdas nos treinos da etapa de *k-fold*. Isto evita *overfitting* por treinamento excessivo. A Figura 20 mostra a evolução da acurácia durante o treinamento com todos os folds para a rede CNN M, em que o número de épocas foi definido pela metodologia como 66.

Figura 20 – Evolução da acurácia durante o treino da rede CNN A.



Fonte: O autor

Neste contexto, a CNN N apresentou um tempo de execução total menor que a metade da melhor rede. A principal diferença entre as duas redes é o tamanho das janelas de *pooling*. A CNN M tem janelas de [[1, 2], [2, 1], [1, 2]], enquanto a CNN N tem janelas de [[1,10], [20,1]]. Como indicado na seção II, as janelas de *polling* são úteis para diminuir as dimensões dos mapas de características das camadas de convoluções, além de permitir que *kernels* pequenos possam identificar parâmetros com grandes extensões nas imagens. Assim, ao diminuir as dimensões dos dados, tais camadas possibilitam um menor tempo de treino.

Outro aspecto de destaque, é que a CNN A é a mesma configuração que obteve o melhor resultado para a classificação em figuras, apresentada na seção 5.1. Como mostrado na tabela, na classificação por notas, o modelo apresentou apenas 75,8% de acurácia para o conjunto de

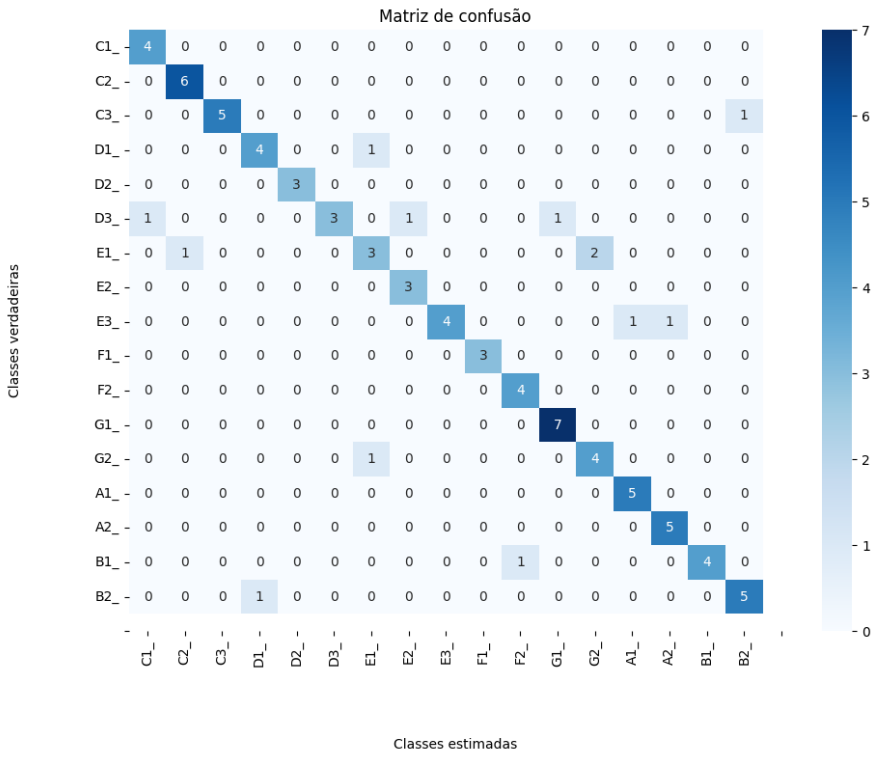
teste, no treino com todos os *folds*. Isto mostra como os dois problemas de classificação analisados neste trabalho têm características diferentes, exigindo configurações também distintas para se obter melhores resultados.

A Figuras 21, 22 e 23 mostram as matrizes de confusão para a rede CNN M, considerando respectivamente: a predição para o último *fold*; a predição para o conjunto de teste, com os pesos do treino do último *fold*; e a predição para o conjunto de testes após o treino com todos os *folds*.

Na classificação por notas, os *datasets* estão mais balanceados, isto é, com números de amostras mais próximos para as diferentes classes, se comparado à classificação por figuras. No entanto, o fato de haver mais classes, faz com que haja, em geral, menos elementos por classe.

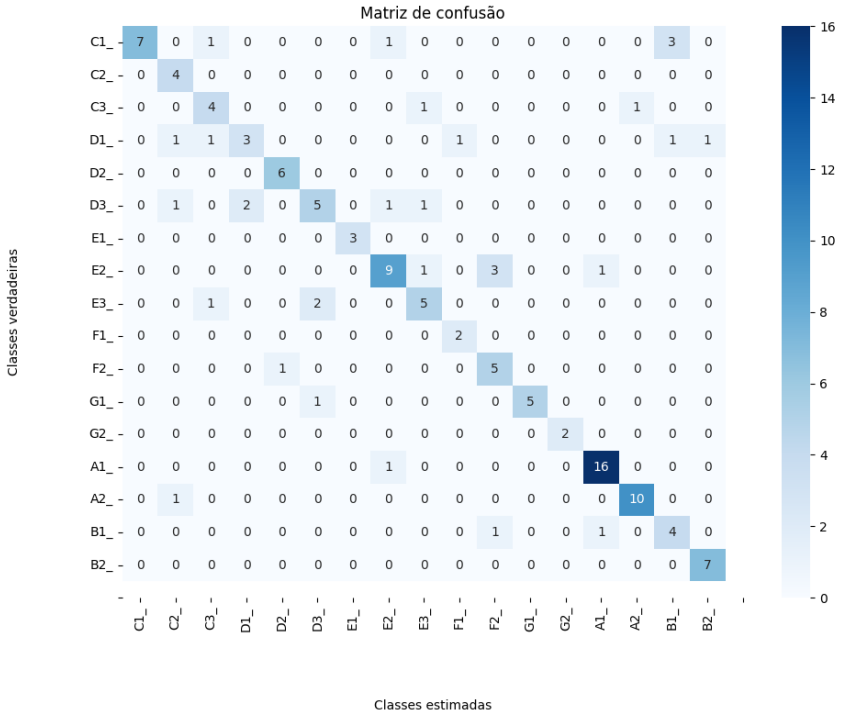
Novamente observa-se uma melhora considerável na predição do conjunto de teste utilizando todos os *folds* no treino (Figura 23). Nesta figura, observa-se que a rede erra duas amostras 'E3', classificados como C3. Neste caso, a proximidade das notas na partitura explica o motivo do erro. A classificação de três amostras 'E2' como 'F2' é outro exemplo de notas próximas confundidas pela rede.

Figura 21 – Matriz de confusão da predição para o último *fold (CNN-M)*.



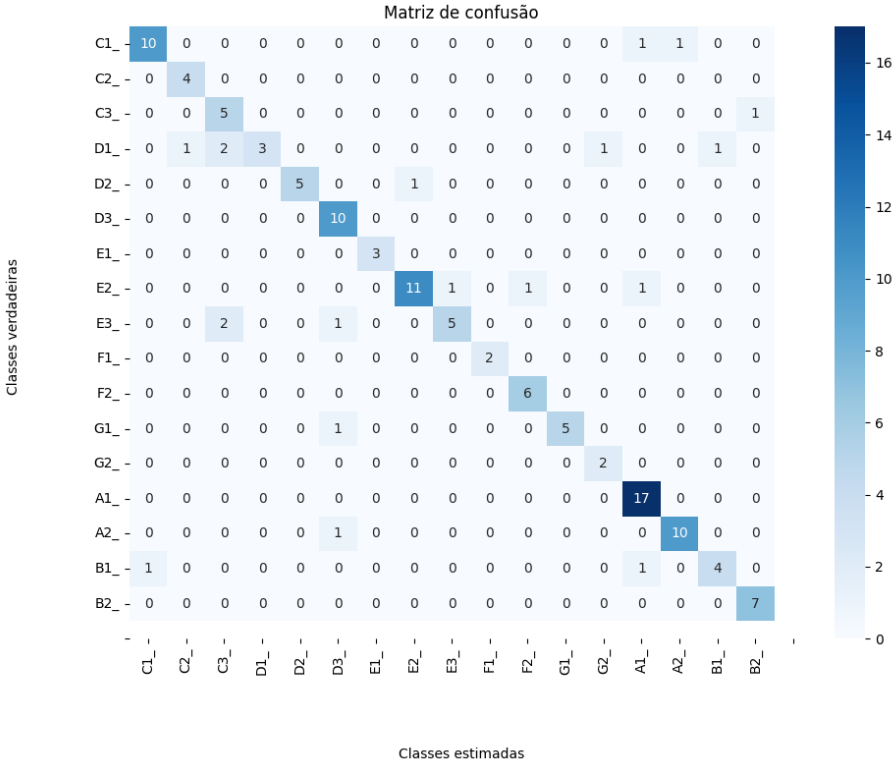
Fonte: O autor

Figura 22 – Matriz de confusão da predição para o conjunto de teste, com os pesos do treino do último fold (CNN-M).



Fonte: O autor

Figura 23 – Matriz de confusão da predição para o conjunto de teste com todos os *folds* para treino (CNN-M).



Fonte: O autor

CAPÍTULO V – CONCLUSÃO E TRABALHOS FUTUROS

Neta seção são apresentadas as conclusões considerando os experimentos e resultados executados, bem como algumas sugestões para trabalhos futuros.

5.2 CONCLUSÃO

Os resultados obtidos neste trabalho foram satisfatórios, visto que o objetivo principal foi cumprido, uma vez que se explorou diversos aspectos relacionados às redes neurais, aplicando-as na classificação de figuras e notas musicais. Os erros de predição encontrados para as redes implementada, não invalida os resultados do trabalho, visto que algumas referências indicam que modelos com elevada complexidade e considerados referência para identificação de elementos em partituras, tiveram dificuldades para identificar hastes de notas, elementos muito comuns em partituras. Isto é, a identificação de elementos em partituras ainda é um campo de pesquisa que carece de melhorias.

Dentre os objetivos específicos, destaca-se que o estudo do sistema de escrita musical em partitura, das técnicas de inteligência artificial com foco em imagens, nomeadamente as redes convolucionais, bem como de recursos e bibliotecas da linguagem Python, utilizados para implementar as redes, permitiram que os resultados pudessem ser alcançados.

Também é interessante destacar que foram manipulados diversos parâmetros das redes neurais, o que permitiu, não só verificar as configurações com melhores métricas, mas também analisar como alguns parâmetros influenciam no desempenho das redes.

Assim, o presente trabalho contribui para a estruturação do conhecimento em relação às Redes Neurais e *Deep Learning*. Para isso, utilizou-se diferentes métodos: o *perceptron* multicamadas e a rede neural convolucional. Assim, representa um alicerce para futuros trabalhos em diferentes áreas, principalmente relacionados às redes neurais e problemas de classificação.

5.3 TRABALHOS FUTUROS

Este trabalho permitiu identificar que há diversas possibilidades para trabalhos futuros. Uma análise mais profunda dos parâmetros dos modelos e seus valores ótimos poderá ser realizada tendo como base os códigos implementados neste trabalho, uma vez que foram feitos de maneira flexível para o teste de diferentes configurações de parâmetros.

Dentre as possibilidades para futuros, também temos a implementação da detecção e classificação de acordes, visto que a proposta inicial foi simplificada para identificar notas representadas individualmente. A segmentação de imagens, também é um seguimento de estudo importante, visto que pode permitir a digitalização de partituras completas. Neste sentido, também pode-se realizar a implementação de um *software* que utilize *deep learning* para identificação em tempo real, de partituras escritas por máquina e, em uma evolução, de partituras escritas à mão.

A aplicação de redes neurais em outros relacionados à música ou não também são outra opção para pesquisas, visto que esses algoritmos são flexíveis. Neste sentido, a aplicação de redes neurais para a classificação e separação de sons de instrumentos e vozes também representa uma alternativa interessante.

CAPÍTULO VI – REFERÊNCIAS

Agarwal; Varun; Prabakeran. OMR Reader Info Scanner. **9th International Conference on Advanced Computing and Communication Systems (ICACCS)**, Coimbatore, India, 2023., p. 205-209

Aggarwal. **Neural Networks and Deep Learning**. [S.l.]: Springer, 2018.

Arobas Music. Music notation software. **Guitar Pro**. Disponivel em: https://www.guitar-pro.com/10-products. Acesso em: 18 Janeiro 2024.

Babini; Marranghello. **Introdução às Redes Neurais Artificiais**. São José do Rio Preto - SP: UNESP, 2007.

Baldo; Rezende. Analise dos filtros não-lineares. **Disciplina de Introdução ao Processamento de Imagens Digitais**, 2004. Disponivel em: https://www.ic.unicamp.br/~rocha/msc/ipdi/filtros_nao_lineares.pdf. Acesso em: 3 jan. 2024.

Barcellos. Claves musicais e armaduras de claves. **Educação Musical**, 2014. Disponivel em: https://juarezbarcellos.com/2014/05/13/claves-musicais-e-armaduras-de-claves/. Acesso em: 28 dez. 2023.

Baró *et al.* From Optical Music Recognition to Handwritten Music Recognition: A baseline. **Pattern Recognition Letters**, 26 fev. 2019.

Barreto. A Importância da Musicalização na Educação Infantil e no Ensino Fundamental. **Música Sacra e Adoração**, 2005.

Bellini; Bruno; Nesi. Optical Music Sheet Segmentation. **Dep. of Systems and Informatic University of Florence**, 07 ago. 2022., p. 8

Calvo-Zaragoza; Jr.; Pacha. Understanding Optical Music Recognition. **ACM Computing Surveys**, New York, NY, USA, 53, 2020., p. 1-35

Chollet. **Deep Learning with Python, Second Edition**. [S.l.]: Manning, 2021.

Chuanzhen Li; Juanjuan Cai. Optical Music Notes Recognition for Printed Music Score. **International Symposium on Computational Intelligence and Design**, 2018.

Cozman; Plonski; Neri. **Inteligência Artificial:** Avanços e Tendências. São Paulo: Instituto de Estudos Avançados, 2021. Disponivel em: https://d1wqtxts1xzle7.cloudfront.net/69708200/Livro_Inteligencia_Artificial-libre.pdf?1631725915=&response-content-disposition=inline%3B+filename%3DNovas_questoes_para_sociologia_contempor.pdf&Expires=1700854413&Signature=EQlNoihCcHSivLcQmazZlYOm~xNY6Y6r.

Fukushima. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. **Biological Cybernetics**, v. 36, p. 193-202, 1980.

Géron. **Hands-On Machine Learning with Scikit-Learn & TensorFlow**. Gravenstein Highway North, California, EUA: O'Reilly Media, 2017.

GLOBAL Music Report 2023. **International Federation of the Phonographyc Industry**, 2023. Disponivel em: https://globalmusicreport.ifpi.org. Acesso em: 14 jan. 2024.

Goto. Física e música em consonância. **Revista Brasileira de Ensino de Física**, p. 8, 2009.

Huang *et al.* Deep Learning-Based Sum Data Rate and Energy Efficiency Optimization for MIMO-NOMA Systems. **IEEE Transactions on Wireless Communications**, v. 19, ago. 2020. ISSN 10.1109/TWC.2020.2992786.

Jimit K. Shah. SangCTC-Improving Traditional CTC Loss. **2019 IEEE Pune Section International Conference**, Pune, India, 18-20 dez. 2019.

Kingma; Ba. **Adam:** A Method for Stochastic Optimization. International Conference on Learning Representations. San Diego: ICLR. 2014.

LeCun *et al.* Backpropagation Applied to Handwritten Zip Code Recognition. **Neural Computation**, v. 1(4), p. 541-551, 1989.

Manage *et al.* An Intelligent Text Reader based on Python. **3rd International Conference on Intelligent Sustainable Systems**, 2020. Disponivel em: https://ieeexplore-ieee-org.ez34.periodicos.capes.gov.br/document/9315996. Acesso em: 24 nov. 2023.

Mannis. Processamento técnico de partituras e registros sonoros: de AACR2 a FRBR e RDA. **4º Congresso Brasileiro de Iconografia Musical & 2º Congresso Brasileiro de Pesquisa e Sistemas de Informação em Música**, 2017.

Mao; Mohri; Zhong. **Cross-Entropy Loss Functions:** Theoretical Analysis and Applications. ICML'23: Proceedings of the 40th International Conference on Machine Learning. Honolulu, Hawaii, USA: JMLR.org. 2023.

Nyakundi. Why Python is Good for Beginners – and How to Start Learning It. **freeCodeCamp**, 2023. Disponivel em: https://www.freecodecamp.org/news/why-learn-python-and-how-to-get-started/#:~:text=Python%20is%20a%20programming%20language,tasks%20and%20in%20different%20industries. Acesso em: 18 Janeiro 2024.

Oliveira; Casagrande; Galerani. A Evolução Tecnológica E Sua Influência Na Educação. **Revista Interface Tecnológica**, São Paulo, 2016. Disponivel em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/123. Acesso em: 24 nov 2023.

Peters. PEP 20 – The Zen of Python. **Python Enhancement Proposals**, 2004. Disponivel em: https://peps.python.org/pep-0020/. Acesso em: 18 Janeiro 2024.

Rebelo *et al.* Optical music recognition: state-of-the-art and open issues. **TRENDS AND SURVEYS**, 02 mar. 2012., p. 18

Santos. Classificação de Padrões de Imagens: Função Objetivo para Perceptron Multicamada e Máquina de Aprendizado Extremo Convolucional, Recife, 2019.

Schmeling. **Berklee - Teoria da Música**. São Paulo: Passarim, 2015.

Sérvio. O que é e para que serve o Python? **Olhar Digial**, 2021. Disponivel em: https://olhardigital.com.br/2021/10/04/tira-duvidas/o-que-e-para-que-serve-o-python/. Acesso em: 08 nov. 2022.

Shatri; Fazekas. Optical Music Recognition: State Of The Art And Major, 22 jun. 2020., p. 10

Shibata; Ikeda. **Effect of number of hidden neurons on learning in large-scale layered neural networks**. 2009 ICCAS-SICE. Fukuoka, Japan: IEEE. 2009, p. 5008-5013.

Silva. A importância da música nas aulas de geografia: práticas e métodos diferenciados no uso da música como metodologia de ensino nas aulas de geografia., 2015., p. 46

Sousa. A Evolução da Notação Musical do Ocidente na História do Livro até à Invenção da Imprensa. **Universidade Da Beira Interior**, p. 127, 2012.

Sultana; Sufian; Dutta. **Advancements in Image Classification using Convolutional Neural Network**. 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN). [S.l.]: [s.n.]. 2018, p. 122-129.

Tuggener *et al.* **The DeepScoresV2 Dataset and Benchmark for Music Object Detection**. 2020 25th International Conference on Pattern Recognition (ICPR). Milan, Italy: IEEE. 2021, p. 9188-9195.

Waibel *et al.* Phoneme Recognition Using Time-Delay Neural Networks. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, v. 37(3), p. 328-339, 1989.

Wingate. Training and Tuning, 2020. Disponivel em: https://ryanwingate.com/intro-to-machine-learning/supervised/training-and-tuning/. Acesso em: 24 dez. 2023.

Yadav; Shukla. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. **6th Innternational Conference on Advanced Computing**, 2016., p. 6

Zotto. **A Importância Da Música No Processo De Ensino E Aprendizagem**. Universidade Tecnológica Federal do Paraná. Medianeira. 2018.