UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

LEANDRO SEBEN COLLE

DESENVOLVIMENTO DE UM RECONHECEDOR ÓPTICO DE SÍMBOLOS MUSICAIS EM PARTITURAS MONOFÔNICAS OCIDENTAIS

BENTO GONÇALVES

2022

LEANDRO SEBEN COLLE

DESENVOLVIMENTO DE UM RECONHECEDOR ÓPTICO DE SÍMBOLOS MUSICAIS EM PARTITURAS MONOFÔNICAS OCIDENTAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Ricardo Vargas Dorneles

BENTO GONÇALVES

2022

LEANDRO SEBEN COLLE

DESENVOLVIMENTO DE UM RECONHECEDOR ÓPTICO DE SÍMBOLOS MUSICAIS EM PARTITURAS MONOFÔNICAS OCIDENTAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 01/12/2022

BANCA EXAMINADORA

Prof. Dr. Ricardo Vargas Dorneles
Universidade de Caxias do Sul - UCS

Prof. Me. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul - UCS

Prof. Dr. André Luis Martinotto
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

Aos meus pais, João Carlos Colle e Silvana Seben Colle, um agradecimento muito especial pelo grande incentivo na minha trajetória acadêmica e pelo apoio nos momentos difíceis.

Aos meus sogros Gilmar Pedro Benincá e Claudete Helena Benincá, pela paciência e ajuda em momentos de muito trabalho e exaustão.

Ao professor Dr. Ricardo Vargas Dorneles, pela sua paciência e orientação, sempre disponível e prestativo em momentos de dúvidas.

Aos professores participantes da banca Me. Alexandre Erasmo Krohn Nascimento e Dr. André Luis Martinotto, pelas opiniões e críticas, que influenciaram de forma positiva na busca de ideias e melhorias.

E um agradecimento muito especial a minha namorada, Priscila Benincá, que sempre esteve ao meu lado nas etapas mais marcantes e decisivas da minha trajetória, sempre me apoiando e incentivando a nunca desistir.

Muito Obrigado.

"O sábio nunca diz tudo o que pensa, mas sempre pensa tudo o que diz."

Aristóteles

RESUMO

O reconhecimento óptico de música é a área da computação que estuda formas de identificar e interpretar imagens de símbolos musicais em partituras. A partir da interpretação desses símbolos, é possível estender o estudo para o campo da análise ou da aplicação. Existem projetos que utilizam o reconhecimento óptico de música para transformar as informações visuais das partituras em arquivos musicais de formato MIDI. Por outro lado, existem aplicações que convertem essas informações em formatos mais estruturados, como o MusicXML, possibilitando um trabalho de análise mais amplo, até mesmo para identificação de padrões de composição. Este projeto acadêmico teve como objetivo desenvolver um reconhecedor óptico de partituras monofônicas ocidentais, utilizando técnicas de processamento e classificação de imagens, restringindo-se a alguns dos símbolos musicais existentes e codificando a representação final no formato MIDI. Para o pré-processamento da imagem foram utilizados os métodos de binarização global e de Otsu, o filtro Gaussiano e operações morfológicas de erosão e dilatação. Para a classificação dos símbolos foi adotado um modelo treinado através das técnicas de rede neural convolucional em conjunto com uma rede neural recorrente, utilizando o *dataset* PRIMUS. Para a predição, além do *dataset* PRIMUS, foram utilizadas pautas próprias para mais validações e testes de classificação. As pautas próprias foram geradas utilizando o sistema Web Noteflight. O protótipo retornou melhores resultados utilizando a binarização de Otsu em conjunto com o filtro Gaussiano, com máscara de 3x3, e o parâmetro de rotação mais adequado para o alinhamento das pautas foi de 0,25 graus. O percentual total de símbolos classificados corretamente atingiu 97,43% para as imagens selecionadas do *dataset* PRIMUS e 77,38% para as imagens próprias geradas.

Palavras-chave: Notação Musical. Reconhecimento Óptico de Partitura. Processamento Digital de Imagem. Classificação de Imagem.

ABSTRACT

Optical music recognition is the area of computing that studies ways to identify and interpret musical symbol images of a musical score. From the interpretation of these symbols, it is possible to extend the study into the field of analysis or application. There are projects that use optical music recognition to convert visual information from scores into MIDI format music files. On the other hand, there are applications that convert this information into a more structured format, such as MusicXML, allowing a broader analysis work, even for identification of composition patterns. This academic project aimed to develop an optical recognizer for western monophonic musical scores, focusing on a few known musical symbols, using image processing and classification techniques, encoding the final representation into MIDI format. For image preprocessing, the global and Otsu binarization methods, the Gaussian filter and morphological erosion and dilation operations were used. For the classification of the symbols, it was adopted a model trained through the convolutional neural network techniques in conjunction with a recurrent neural network, using the PRIMUS dataset. For prediction, in addition to the PRIMUS dataset, proprietary staves were used for further validation and classification tests. The own staves were generated using the Noteflight web system. The prototype returned the best results using the Otsu binarization in conjunction with the Gaussian filter with a 3x3 mask, and the most suitable rotation parameter for the alignment of the staves was 0.25 degrees. The total percentage of correctly classified symbols reached 97.43% for the selected images from the PRIMUS dataset and 77.38% for the generated own images.

Keywords: Musical Notation. Optical Music Recognition. Digital Image Processing. Image Classification.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*

BFM *Brute Force Matcher*

BOK *Bag of KeyPoints*

CNN *Convolutional Neural Network*

CRNN *Recurrent Neural Network*

CTC *Connectionist Temporal Classification*

CWMN *Common Western Music Notation*

FFN *Feed-Forward Network*

GPU *Graphics Processing Unit*

HOMUS *Handwritten Online Musical Symbols*

HMM *Hidden Markov Models*

IBL *Instance-Based Learning*

KNN *K-Nearest Neighbour*

LTH *Line Track Height*

MEI *Music Encoding Initiative*

MLP *Multi-Layer Perceptron*

MIDI *Musical Instrument Digital Interface*

NN *Neural Network*

OCR *Optical Character Recognition*

OMR *Optical Music Recognition*

ORB *Oriented FAST and Rotated BRIEF*

PRIMUS *Printed Images of Music Staves*

RGB *Red Green Blue*

RLE *Run-Length Encoding*

RNN *Recurrent Neural Network*

SVM *Support Vector Machines*

SLH *Staffline Height*

SLSH *Staffline Space Height*

SSH *Staffspace Height*

XML *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

As partituras são compostas por símbolos que representam elementos musicais: nota, tonalidade e tempo. Esses elementos juntos compõem a semântica da música. Ao longo dos anos, a música foi tradicionalmente escrita com tinta e papel. Durante a década de 1980, foram desenvolvidos os primeiros programas de composição de música por computador, que revolucionaram a maneira como a música pode ser gravada. Atualmente, a abordagem mais comum para transformar dados de música em um formato legível para o computador combina entrada de teclado musical com teclado e mouse de computador (NOVOTNý; POKORNý, 2015). Nesta forma de conversão, que é um procedimento demorado e requer habilidade no instrumento, o teclado musical é usado para inserir as notas e o teclado do computador é usado para corrigir possíveis erros ou adicionar mais informações pertinentes à música.

Contudo, as representações musicais em papel podem ser entendidas por um sistema de *Optical Music Recognition* (OMR), que, de forma geral, as interpreta através da identificação e classificação dos símbolos, codificando o significado contido na imagem da partitura. O objetivo principal de sistemas como esse é efetuar a interpretação de forma automática, representando os resultados do reconhecimento em formatos digitais, que possibilitam o armazenamento da informação musical. Desta forma, é gerada uma oportunidade de preservação musical e cultural, além da possibilidade de utilização da informação para diferentes aplicações, como execução e edição de música.

É comum encontrar, ainda hoje, partituras em papel e até mesmo escritas à mão. Músicos contemporâneos podem preferir a escrita manual para registrar suas ideias de composição. Entretanto, um sistema OMR pode simplificar a aquisição de dados musicais e otimizar o tempo de um processo antes realizado manualmente.

O reconhecimento de partituras manuscritas pode ser complexo e apresentar grandes desafios devido à ampla variação de caligrafias, resultando em diferentes formas de desenho para um mesmo símbolo. Além disso, existe a falta de padrão na estruturação da pauta, pois compositores podem omitir símbolos propositalmente, desobedecendo regras de notação com o objetivo de simplificar a escrita musical.

Tendo em vista os desafios encontrados na área, este trabalho realiza o estudo das técnicas utilizadas para o processamento de imagens e reconhecimento de pautas musicais, com a proposta de desenvolver um reconhecedor de partituras automático e eficiente. O estudo não objetiva alcançar uma solução para um reconhecimento completo de pautas musicais, restringindo-se apenas às imagens de partituras monofônicas digitais e à identificação de símbolos que resultem em um reconhecimento robusto e confiável.

1.1 OBJETIVO GERAL

Este projeto teve como objetivo geral o desenvolvimento de um reconhecedor óptico de símbolos musicais em partituras monofônicas ocidentais.

1.2 OBJETIVOS ESPECÍFICOS

1. Definição da arquitetura de um sistema OMR;
2. Desenvolvimento de um sistema OMR.

1.3 JUSTIFICATIVA

Utilizar a tecnologia para desenvolver ferramentas de auxílio é uma forma significativa para agregar qualidade a processos manuais, tornando-os mais simplificados e trazendo agilidade para as pessoas envolvidas. O objetivo de desenvolver um reconhecedor de símbolos musicais foi gerar arquivos de áudio para contribuir com a área da música, auxiliando na interpretação e leitura de partituras através do sistema auditivo humano.

As partituras escritas digitalmente e impressas em papel podem ser utilizadas como referência para estudo e execução de músicas. Além disso, o manuseio do papel para anotações ou viradas de página pode ser considerado mais fácil para algumas pessoas do que a utilização de periféricos, como mouse e teclado, que podem tornar o processo mais lento. Considerando estes fatos, um sistema capaz de reconhecer os principais símbolos musicais pode auxiliar em momentos de leitura de partituras e entendimento da música.

1.4 ESTRUTURA DO TRABALHO

Este trabalho é composto por 6 capítulos posteriores à introdução.

O capítulo 2 apresenta uma revisão bibliográfica dos assuntos considerados importantes para a compreensão deste trabalho. Nesse capítulo são abordados os conceitos de notação musical e é apresentada a estrutura de uma partitura e seu conjunto de símbolos musicais. Os diferentes níveis e formatos de partituras são exemplificados e brevemente descritos. O conceito de OMR é apresentado e sua arquitetura é subdividida em diferentes tópicos, que são descritos e exemplificados com base em pesquisas feitas em outros trabalhos (REBELO *et al.*, 2012), (SHATRI; FAZEKAS, 2020).

O capítulo 3 apresenta alguns trabalhos relacionados à área de processamento e reconhecimento de partituras, com diferentes abordagens de classificação de imagens. As principais metodologias utilizadas e os resultados obtidos são brevemente descritos, com base nas especificações e conclusões de cada autor.

O capítulo 4 é destinado à introdução e caracterização, baseando-se em critérios selecionados, dos *datasets* existentes e das bibliotecas utilizadas neste projeto. Esse capítulo envolve a escolha das ferramentas mais adequadas para a solução desenvolvida.

O capítulo 5 descreve o sistema desenvolvido para atingir o objetivo deste trabalho. Além disso, o capítulo aborda os detalhes da arquitetura, das ferramentas e a metodologia utilizada para a implementação do reconhecedor.

O capítulo 6 apresenta as análises feitas sobre os resultados obtidos com a execução do sistema de reconhecimento, considerando diferentes parâmetros de entrada.

O capítulo 7 é destinado às conclusões do projeto. Nesse capítulo as dificuldades encontradas são descritas, a visão geral em torno dos resultados obtidos é destacada e são levantadas sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

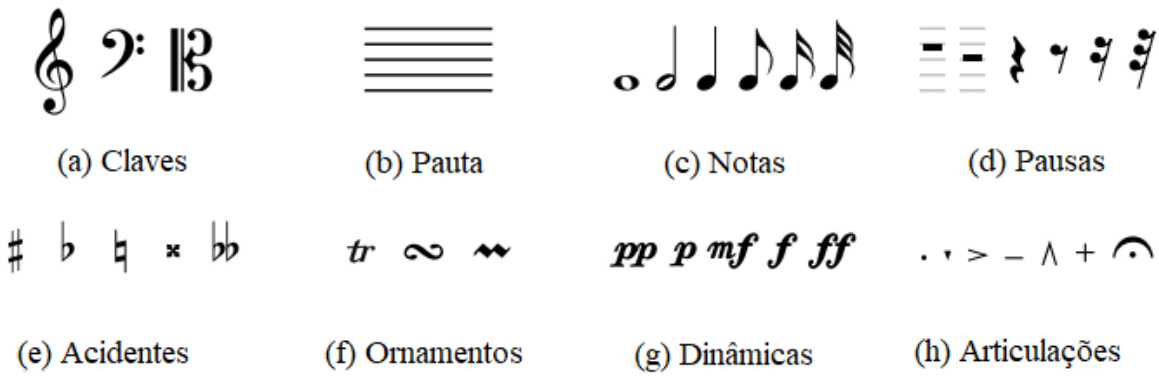
Este capítulo tem o objetivo de apresentar os tópicos conceituais e teóricos deste trabalho, abordando os assuntos necessários para o entendimento do projeto.

2.1 NOTAÇÃO MUSICAL

Notação musical refere-se a um grupo de sistemas de escrita com os quais uma vasta gama de músicas podem ser visualmente codificadas para que músicos possam executá-las mais tarde (CALVO-ZARAGOZA; JR.; PACHA, 2020). Em outras palavras, é um sistema de escrita musical universal. Este padrão de escrita pode representar um compasso, trecho ou até mesmo uma peça musical inteira. Existem outros tipos de notação musical (CALVO-ZARAGOZA *et al.*, 2014), porém, neste trabalho, o foco será o reconhecimento de partituras ocidentais, também denominadas como *Common Western Music Notation* (CWMN).

Cada tipo de notação musical possui sua própria linguagem e estrutura simbólica. No caso das CWMN, existem quatro propriedades: tonalidade, duração, volume e timbre. Na Figura 1 são ilustrados os símbolos musicais que pertencem a essa notação. Uma clave (Figura 1a) determina a tonalidade de cada linha e espaço da pauta (Figura 1b). Acidentes (Figura 1e) modificam a altura (tonalidade) da nota, porém, a tonalidade original de cada nota é definida por seu posicionamento vertical na pauta, onde sua duração relativa é definida por sua figura de nota (Figura 1c). Ornamentos (Figura 1f) são alterações que acrescentam algum embelezamento em cada nota individualmente. Pausas (Figura 1d) são os símbolos que representam a ausência de som na música, isto é, representam a duração do silêncio. Dinâmicas (Figura 1g) são as figuras que representam a variação do volume. Por fim, as articulações (Figura 1h) alteram o timbre ou a duração de uma nota.

Figura 1 – Símbolos musicais da notação CWMN.



Fonte: Adaptado de Novotný e Pokorný (2015)

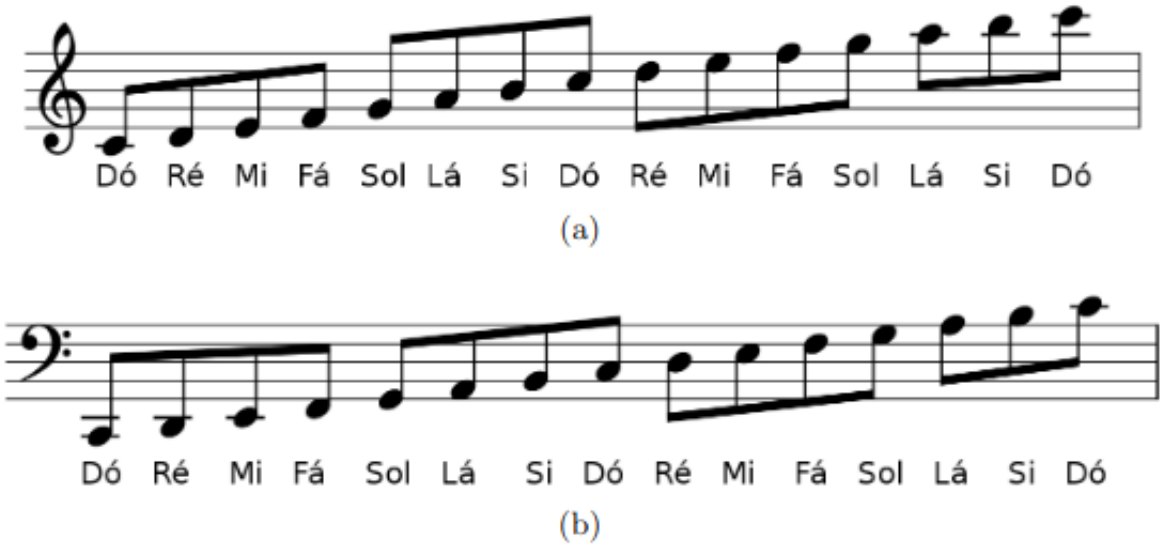
Conforme explicado anteriormente, a tonalidade original das notas é definida pelo seu posicionamento na pauta, mas as mesmas notas também são representadas por letras do alfabeto. As sete notas musicais são: Dó, Ré, Mi, Fá, Sol, Lá e Si. Suas representações alfabéticas são exemplificadas na Tabela 1 e seus posicionamentos para as claves de Sol e Fá são demonstrados na Figura 2.

Tabela 1 – As sete notas musicais.

Nota	**Representação no alfabeto**
Dó	C
Ré	D
Mi	E
Fá	F
Sol	G
Lá	A
Si	B

Fonte: O Autor (2022)

Figura 2 – Clave de sol e suas notas (a) e clave de fá e suas notas (b).



Fonte: Adaptado de Furukawa e Pedrini (2021)

Além de serem compostas por diversos símbolos, as partituras podem ser classificadas por diferentes complexidades. De acordo com Byrd e Simonsen (2015), existem quatro categorias, conforme ilustra a Figura 3:

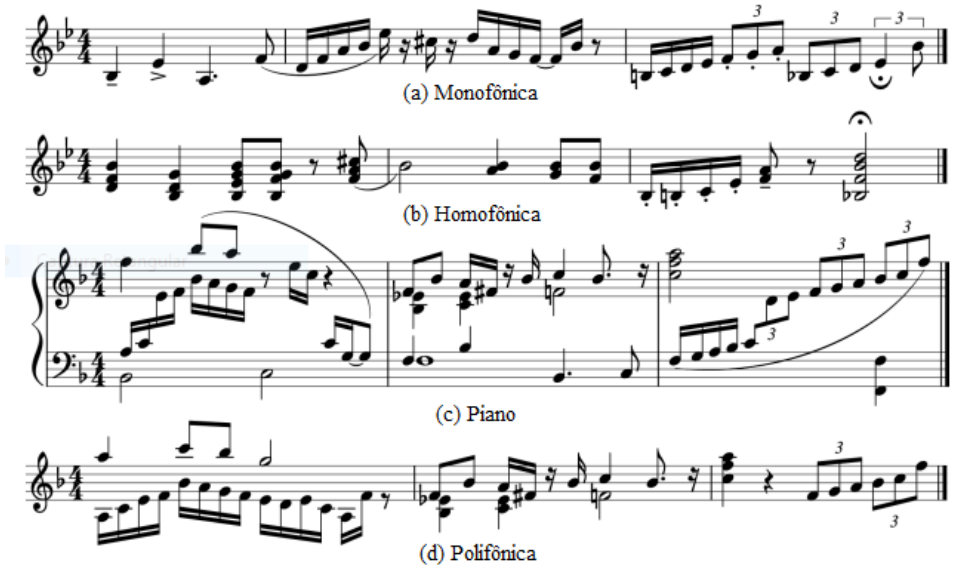
a) Monofônica: música em uma única pauta com apenas uma nota por tempo;

b) Homofônica: música com acordes, porém apenas uma voz;

c) Piano: música em múltiplas pautas com interação de múltiplas vozes;

d) Polifônica: música com múltiplas vozes em apenas uma pauta;

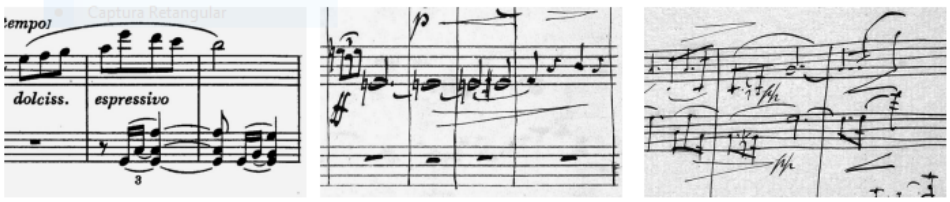
Figura 3 – Exemplos de diferentes complexidades de partituras.



Fonte: Adaptado de Byrd e Simonsen (2015)

As partituras são o objeto central do sistema de reconhecimento, e, conforme Novotný e Pokorný (2015), para efeitos de reconhecimento musical, as imagens de partituras podem ser divididas em três categorias: partitura inteiramente impressa (Figura 4a), partitura manuscrita sobre as linhas de pauta pré-impressas (Figura 4b) e partitura inteiramente manuscrita (Figura 4c).

Figura 4 – Exemplos de diferentes formas de imagens de partituras.



(a) Totalmente impressa (b) Linhas de pauta pré-impressa (c) Totalmente escrita a mão

Fonte: Adaptado de Novotný e Pokorný (2015)

Tendo em vista as características de notação musical, pode-se definir um objetivo de reconhecimento mais abrangente ou mais restrito. Essa escolha envolve as diferentes complexidades de partituras, suas diferentes formas de imagem e o conjunto de símbolos. É comum encontrar trabalhos que reconhecem imagens de partituras totalmente impressas (MAURICENZ, 2013), (MARGARIDA; SILVA, 2013), (FURUKAWA; PEDRINI, 2021), pois é a forma de imagem mais bem definida para o reconhecimento. Entretanto, existem estudos voltados para o reconhecimento de imagens totalmente manuscritas (NG *et al.*, 1999), (SILVA, 2017), (GOECKE, 2003).

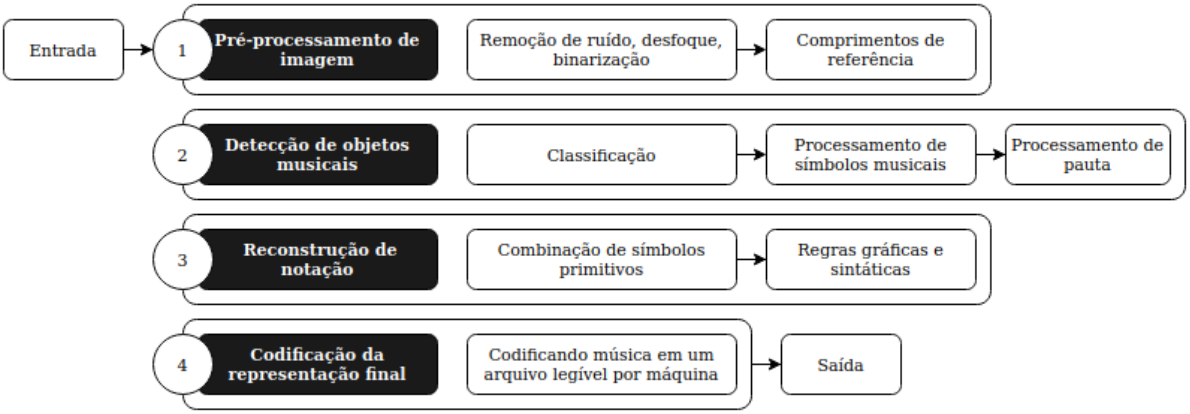
2.2 CONCEITO DE OMR (*OPTICAL MUSIC RECOGNITION*)

O reconhecimento óptico de música pode ser definido de diferentes formas (CALVO-ZARAGOZA; JR.; PACHA, 2020), mas o conceito básico consiste em uma ferramenta computacional que possibilita a extração de informações musicais de imagens de partituras, codificando os símbolos musicais em formatos legíveis para um computador.

2.3 ARQUITETURA DE UM SISTEMA OMR

É fundamental que um sistema de reconhecimento seja subdividido em etapas, assim, cada processo pode ser entendido de maneira detalhada e desenvolvido de forma mais segmentada. A maioria dos autores analisados segue um consenso na subdivisão das tarefas. Neste trabalho, utilizou-se a abordagem mais comum analisada nos estudos prévios (REBELO *et al.*, 2012), (SHATRI; FAZEKAS, 2020), (NOVOTNý; POKORNý, 2015), conforme ilustrado na Figura 5.

Figura 5 – Arquitetura de um sistema de processamento OMR.



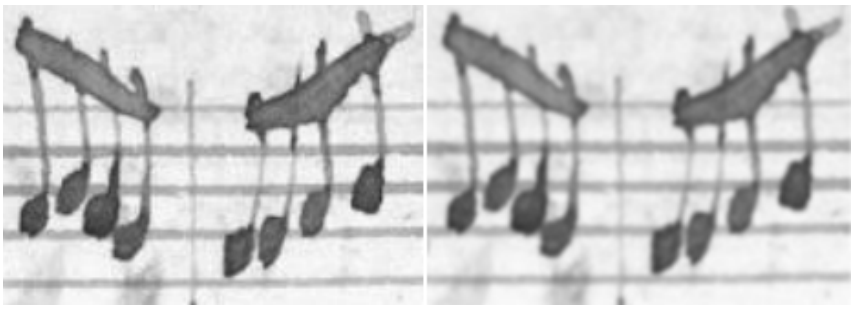
Fonte: Adaptado de Shatri e Fazekas (2020)

2.3.1 Pré-processamento de imagem

Segundo Chaki e Dey (2018), o pré-processamento é um método para transformar uma imagem em sua forma mais limpa, gerando um resultado de saída com menos ruído. Para o autor, o pré-processamento também pode ser utilizado para completar ou ajustar informações inconsistentes em uma imagem, sendo uma etapa fundamental para redução de erros nos processos de segmentação. Abaixo são descritas as operações mais comuns para manipulação de imagens, considerando alguns dos autores estudados (REBELO *et al.*, 2012), (SHATRI; FAZEKAS, 2020):

Filtro Gaussiano é um método muito comum para a eliminação de ruídos em imagens. Com esse filtro o ruído é suavizado, porém, os *pixels* da imagem são deslocados e o sinal fica distorcido, causando um efeito de desfoque (JESUS; JR., 2015), conforme ilustra a Figura 6.

Figura 6 – Aplicação do filtro Gaussiano com máscara de 5x5: antes (esquerda) e depois (direita).

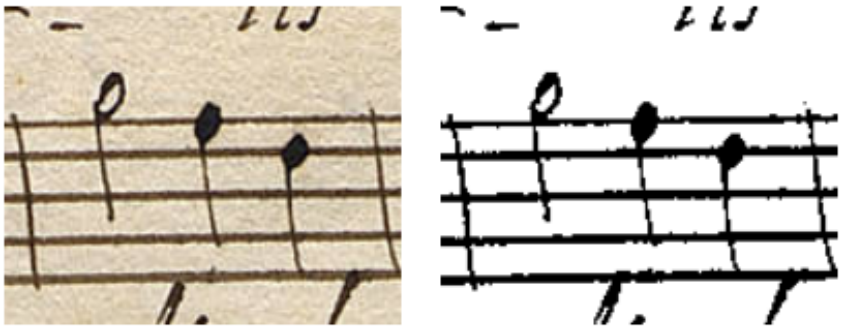


Fonte: O Autor (2022)

Binarização é o procedimento mais comum do pré-processamento em sistemas OMR. Essa operação consiste em converter uma imagem para o formato binário, tornando-a preto e branco. Tecnicamente, uma imagem binária possui apenas dois canais de cores, ao contrário de imagens coloridas, que possuem os canais *Red Green Blue* (RGB) e um canal adicional de brilho, denominado canal Alfa. Além disso, imagens binárias possuem apenas dois planos bem definidos (primeiro plano e plano de fundo), com destaque nos objetos. Esse processo reduz o volume de informações passadas para as etapas seguintes do reconhecimento (segmentação e classificação).

A técnica de binarização mais comum utiliza o conceito de *global threshold*. Essa técnica não demanda um conhecimento especial sobre o conteúdo da imagem que está sendo processada, pois ela aplica um único *threshold* para todo processo de binarização, isto é, toda imagem é binarizada em cima do mesmo limiar. Entretanto, existem técnicas mais avançadas que se baseiam no conhecimento prévio do conteúdo da imagem, chamadas de *binarization based on domain knowledge*. Conforme artigo de Pinto *et al.* (2011), esta técnica aplica um *threshold* para um ou mais *pixels* selecionados, usando informações dos *pixels* vizinhos como parâmetro para melhorar o resultado da binarização e tornar possível a eliminação de rabiscos uniformes da imagem, porém, consumindo maior tempo de processamento.

Figura 7 – Exemplo de binarização: imagem original (esquerda) e imagem binarizada pelo algoritmo de Otsu (direita).

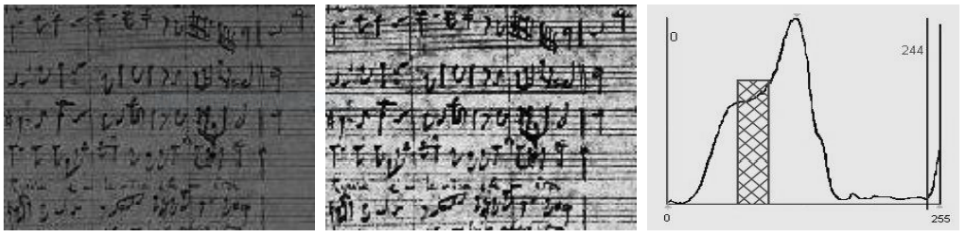


Fonte: Adaptado de Furukawa e Pedrini (2021)

Equalização de histograma é uma técnica usada para a melhoria de contraste de uma imagem. Esse procedimento redistribui os níveis de cinza com o objetivo de gerar uma imagem com um histograma uniforme, entretanto, o brilho da imagem não é preservado (FILHO; NETO, 1999).

Conforme Goecke (2003), idealmente, o histograma de uma imagem deve ter dois picos distintos de valores de frequência, um para o primeiro plano e outro para o plano de fundo. Entretanto, não é o que acontece no exemplo da Figura 8, em que a intensidade dos dois planos se sobrepõem. A área marcada remete à intensidade dos símbolos musicais e a imagem da esquerda e centro mostram o efeito da equalização. Pode-se perceber uma melhora no contraste (imagem do centro) que pode auxiliar no processo de segmentação da imagem.

Figura 8 – Equalização do histograma: antes (esquerda), depois (centro) e histograma antes da equalização (direita).



Fonte: Adaptado de Goecke (2003)

Operações morfológicas de erosão e dilatação são outros procedimentos que, quando utilizados em conjunto, podem remover pequenos ruídos em imagens sem degradar os objetos importantes. Conforme Goecke (2003), essas duas técnicas também podem ser utilizadas para a remoção do ruído *"salt and pepper"*, ilustrado na Figura 9.

Figura 9 – Imagem com ruído (esquerda) e imagem filtrada (direita).

Fonte: O Autor (2022)

O uso da operação de erosão diminui as características dos objetos e também pode separá-los caso estejam conectados. Já a operação de dilatação efetua o aumento da área dos objetos, acentuando suas características (FILHO; NETO, 1999). O uso dessas técnicas em sequência (erosão - dilatação) efetua o encolhimento dos objetos, removendo pequenos ruídos da imagem, e acentua as características dos objetos novamente. Os ruídos são apagados na primeira filtragem e não retornam novamente.

2.3.2 Reconhecimento de símbolos musicais

Após o pré-processamento existe a etapa de reconhecimento. Essa parte envolve a segmentação e a classificação dos símbolos musicais, podendo ser dividida em três passos, conforme Shatri e Fazekas (2020): processamento da pauta, processamento dos símbolos musicais e, por fim, a classificação.

O **processamento da pauta** tem como objetivo a detecção e remoção de suas linhas. Conforme explicado em seções anteriores, a pauta é composta por um conjunto de cinco linhas horizontais, que subdividem as diferentes figuras de notas em suas respectivas tonalidades. Após a detecção, o próximo passo é calcular duas referências importantes: a espessura das linhas e a distâncias entre as linhas da pauta, conforme demonstra a Figura 10. Essas referências são importantes para os passos seguintes do reconhecimento, tais como a dedução do tamanho e posicionamento das figuras de notas.

Figura 10 – Exemplo de espessura da linha e distância entre linhas da pauta.

Fonte: Adaptado de Cardoso e Rebelo (2010)

Neste trabalho, estas referências são descritas, abreviadamente, como *Staffline Height* (SLH) e *Staffspace Height* (SSH), mas também podem ser definidas como comprimentos de referência. Levando em consideração os aspectos descritos, Cardoso e Rebelo (2010) inserem o contexto de dois métodos utilizados para a estimativa do SLH e do SSH:

a) **Estimativa convencional**: a maneira mais comum para calcular a dimensão das linhas e espaços é através da codificação *Run-Length Encoding* (RLE). Essa codificação possui uma forma simples de compressão, de modo que ocorrências consecutivas de valores iguais são somadas e colocadas em sequência. No caso de uma imagem binária, essa codificação se torna ainda mais simples, pois é composta por apenas dois valores possíveis: 0 e 1.

   Conforme o exemplo colocado por Cardoso e Rebelo (2010), a sequência 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 é codificada para 8 3 13 8 2. Vale notar que o valor inicial da sequência é 1, no entanto, a mesma regra se aplica para casos em que o valor inicial é 0.

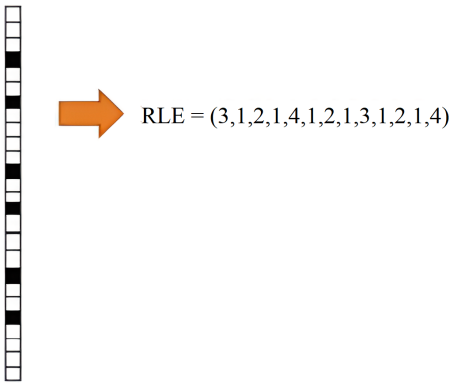
   Conforme essa codificação, as ocorrências pretas (1) mais comuns representam o SLH e as ocorrências brancas (0) mais comuns representam o SSH, conforme exemplo ilustrado na Figura 11.

b) **Estimativa robusta**: embora seja possível alcançar bons resultados com a codificação RLE convencional, ela pode falhar na tentativa de estimar o SLH e o SSH em imagens com qualidade muito baixa. Com isso, Cardoso e Rebelo (2010) introduzem uma abordagem mais robusta para calcular as estimativas.

Para Cardoso e Rebelo (2010), deve-se fazer duas observações: em primeiro lugar, o comprimento da sequência de *pixels* brancos antes e depois dos *pixels* pretos isolados, irá variar muito e aleatoriamente; em segundo lugar, uma alteração no decorrer da espessura do SLH (geralmente devido a ruído) é compensada por uma variação de sinal oposto da distância entre as linhas.

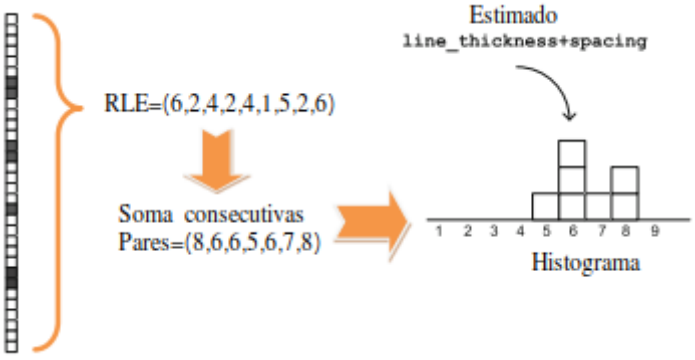
Portanto, conforme Cardoso e Rebelo (2010), a estimativa do SLH e do SSH pode ser mais precisa ao encontrar o valor mais comum da soma das duas linhas verticais consecutivas, isto é, a soma consecutiva de SLH + SSH, ou vice-versa. Através disso, estimativas do SLH e SSH podem ser obtidas de forma mais precisa, sabendo que a soma das duas deve ser igual à altura do *Staffline Space Height* (SLSH), conforme exemplo ilustrado na Figura 12.

Figura 11 – Exemplo da codificação RLE convencional em sequência de *pixels* verticais.



Fonte: Adaptado de Gomez e Sujatha (2017)

Figura 12 – Exemplo da codificação RLE robusta em sequência de *pixels* verticais.

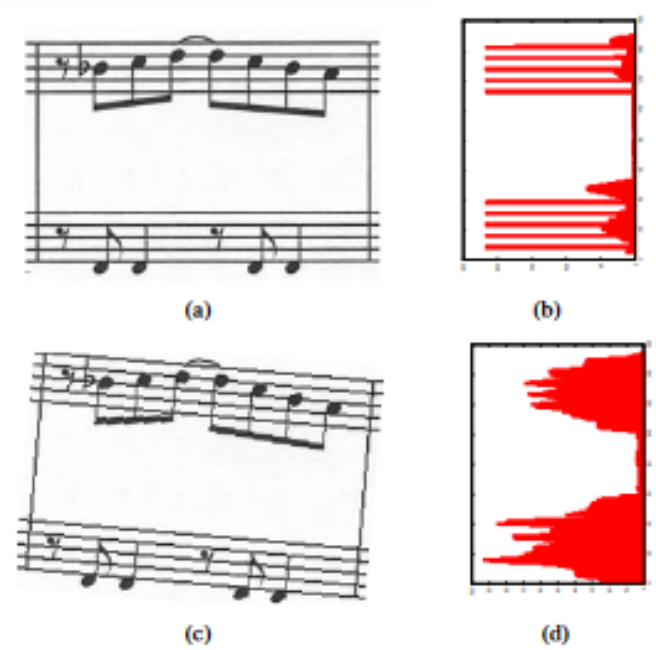


Fonte: Adaptado de Rebelo *et al.* (2012)

Conforme colocado por Cardoso e Rebelo (2010), o método RLE é usado como codificador da partitura para extrair os comprimentos de referência. A codificação da imagem ocorre verticalmente, extraindo o valor mais comum da sequência dos *pixels* pretos e o mais comum da sequência dos *pixels* brancos, representando o SLH e o SSH, respectivamente. Entretanto, considerando uma baixa qualidade da imagem de entrada, essas referências podem ser calculadas erroneamente. Conforme ilustrado anteriormente na Figura 4, enquanto partituras totalmente ou parcialmente digitalizadas podem possuir suas linhas mais paralelas e uniformes, em uma partitura totalmente manuscrita as linhas podem estar mais curvadas e não serem totalmente horizontais, sem contar o nível de ruído que as imagens podem apresentar.

Contudo, é importante que sistemas OMR efetuem um processo de alinhamento de pautas. O ajuste horizontal da imagem garante uma melhor extração dos comprimentos de referência e, consecutivamente, uma melhor classificação dos símbolos musicais. Para isso é possível utilizar a técnica de projeção horizontal, que também pode ser utilizada para a remoção das linhas da pauta. A projeção horizontal de uma imagem binária consiste em um histograma no qual a contagem do número de *pixels* presentes em cada linha da imagem é acumulada e, a partir disso, os comprimentos de referência podem ser obtidos analisando os picos locais da projeção, conforme a Figura 13.

Figura 13 – Partitura horizontal (a) e sua projeção (b). Partitura curvada (c) e sua projeção (d).



Fonte: Adaptado de Margarida e Silva (2013)

Conforme colocado por FuJinaga (1988), se as linhas da pauta são retas e horizontais, consequentemente o histograma possuirá cinco picos distintos, também conhecidos como picos locais ou máximas locais. Na prática, as projeções com o nível mais alto (até cinco) de picos locais representam as imagens de partituras mais horizontais.

Para detecção das linhas da pauta, também é comum utilizar projeções horizontais (FUJINAGA, 1988) (FUJINAGA, 2004), pois elas possibilitam a obtenção das posições das linhas ao encontrar a posição dos picos locais de projeção. Caso a pauta não esteja perfeitamente horizontal, os resultados produzidos podem ser inadequados. Portanto, para alinhar uma pauta horizontalmente, aplicam-se pequenas rotações incrementais na imagem, selecionando a posição daquela com a projeção de maior pico local. Esse alinhamento também é importante para a etapa de remoção das linhas da pauta.

Após a detecção, a remoção das linhas pode ser realizada. Existem diversos métodos que possuem essa finalidade (DALITZ *et al.*, 2008), entretanto, os mais comuns são o já citado RLE e o *Line Track Height* (LTH). O objetivo do LTH é percorrer as linhas da pauta removendo as sequências verticais de *pixels* com tamanho menor que o limiar aplicado, que geralmente é o resultado da soma do SLH+2, conforme ilustrado na Figura 14.

Figura 14 – Exemplo de algoritmo LTH usado na remoção das linhas da pauta (marcação em azul).

Fonte: Adaptado de Gomez e Sujatha (2017)

A ideia do algoritmo vem da observação de que a sequência vertical dos *pixels* dos símbolos musicais são, geralmente, mais longas que as sequências verticais das linhas da pauta. Entretanto, o algoritmo LTH não remove completamente as linhas da pauta quando estas possuem muita variação de altura. Para solucionar isso, Na, Kim e Nquyen (2015) utilizam um método em duas etapas, no qual a primeira etapa consiste na remoção da linha conforme o modo convencional e a segunda etapa percorre a linha novamente, removendo os componentes com um número de *pixels* menor que o limiar aplicado, desta forma, os ruídos gerados inicialmente podem ser eliminados.

Figura 15 – Exemplo de pauta com linhas removidas.

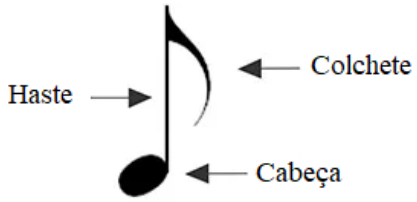
Fonte: Adaptado de Rebelo, Capela e Cardoso (2010)

Apesar das etapas de detecção e remoção das linhas da pauta serem muito utilizadas em sistemas OMR, elas podem ser desconsideradas quando for utilizada a técnica de redes neurais para reconhecimento dos símbolos musicais, considerando que o treino da rede seja feito a partir de imagens contendo as linhas da pauta. Entretanto, a permanência das linhas também pode impactar na classificação dos símbolos de maneira negativa, escondendo características importantes das figuras.

O **processamento dos símbolos musicais**, também chamado de segmentação dos objetos, é a etapa seguinte à detecção e remoção das linhas da pauta. Nesse processo é feito o reconhecimento das figuras, isolando-as, para identificá-las separadamente e construir um modelo de cada objeto. Entretanto, com a evolução da inteligência artificial e a popularização do aprendizado de máquina, o uso de modelos treinados por redes neurais acaba otimizando todo esse processo de reconhecimento e pode trazer melhores resultados na classificação dos símbolos.

Pela forma mais comum de segmentação, é preciso, primeiro, encontrar a posição e o tipo dos símbolos musicais. As figuras são detectadas em suas formas mais primitivas, isto é, em uma nota detecta-se o colchete, haste e cabeça separadamente, ao invés da nota por inteira. O método mais citado nas referências e utilizado para essa segmentação é chamado de decomposição hierárquica.

Figura 16 – Exemplo de primitivas de nota.



Fonte: O Autor (2022)

Rebelo, Capela e Cardoso (2010) utilizam a decomposição hierárquica em seu trabalho, subdividindo os símbolos alvo. Primeiramente, a imagem da partitura é analisada e dividida por pauta. Na sequência, os símbolos são identificados e extraídos, de forma que não ocorram duplicações (considerando as notas conectadas pela barra). O objetivo desta etapa é obter os moldes dos símbolos utilizados para a segmentação. O autor utiliza os comprimentos de referência SLH e SSH, calculados em etapas anteriores, como ponto de partida para a definição do limiar de segmentação.

A **classificação dos símbolos musicais** é a etapa subsequente a segmentação. Nessa parte, os símbolos são classificados com base na sua forma e similaridade, obtidas anteriormente. Conforme passos anteriores, este também possui diferentes abordagens e técnicas, conforme os itens a seguir:

a) ***Hidden Markov Models* (HMM)**: é a técnica menos utilizada em sistemas OMR; teve origem nos estudos e aplicações de *Optical Character Recognition* (OCR), porém, foi adaptada para uso no reconhecimento de símbolos musicais por possibilitar a segmentação e classificação em paralelo.

   Conforme Rebelo, Capela e Cardoso (2010), um HMM é um processo duplamente estocástico, onde o primeiro gera uma sequência de símbolos e o segundo é oculto, podendo ser detectado apenas por meio de outro processo cujas realizações são observáveis. Este processo oculto consiste em um conjunto de estados conectados entre si por uma probabilidade de transição.

b) ***K-Nearest Neighbour* (KNN)**: é uma das técnicas mais simples de aprendizado de máquina, enquadrando-se na categoria de técnicas denominadas *Instance-Based Learning* (IBL). Consiste na extensão da área em torno do ponto de partida, até que um determinado vizinho, definido inicialmente, seja encontrado.

   Rebelo, Capela e Cardoso (2010) explicam que um objeto é classificado por um sistema de votação por maioria, isto é, um símbolo é classificado por seus vizinhos mais próximos e comuns. Para essa classificação, os autores adotam a distância euclidiana.

c) ***Neural Network* (NN)**: é um conjunto de técnicas baseadas no cérebro humano. Essas técnicas consistem no treino de uma rede neural para identificar objetos com base em um *dataset* previamente disponibilizado. Rebelo, Capela e Cardoso (2010) utilizam em seu trabalho uma arquitetura conhecida como *Multi-Layer Perceptron* (MLP), que faz parte do conjunto das *Feed-Forward Network* (FFN).

   Basicamente, um MLP é um conjunto de nós em uma estrutura de camadas. Esses nós também são conhecidos como neurônios e são ligados unidirecionalmente aos neurônios subsequentes.

d) ***Support Vector Machines* (SVM)**: é a técnica que consiste em construir um hiperplano em um espaço N-dimencional (N é o número de características). Desta forma, é feita uma superfície de decisão com grande margem de separação entre as características selecionadas. No estudo de Rebelo, Capela e Cardoso (2010), essa técnica teve uma melhor performance que a HMM.

A fase de classificação visa rotular cada imagem de objeto pelo símbolo nela contido. Conforme Calvo-Zaragoza *et al.* (2014), muitos sistemas OMR dependem de uma extração de características para efetuar a classificação dos símbolos, dependendo das técnicas aplicadas. Essas características são utilizadas para construir um conjunto de amostras que serão usadas no reconhecimento de padrões. A técnica dos descritores de Fourier é bastante eficiente para planos de curvas fechadas e foi utilizada por Mauricenz (2013) em seu trabalho de classificação.

Contudo, considerando o crescimento do uso das redes neurais em muitas aplicações de aprendizado de máquina, estudos mais recentes optam por utilizar esta abordagem na etapa de classificação de sistemas OMR, pois modelos treinados já estão validados e podem apresentar resultados bastante positivos (CALVO-ZARAGOZA; RIZO, 2018) (FURUKAWA; PEDRINI, 2021).

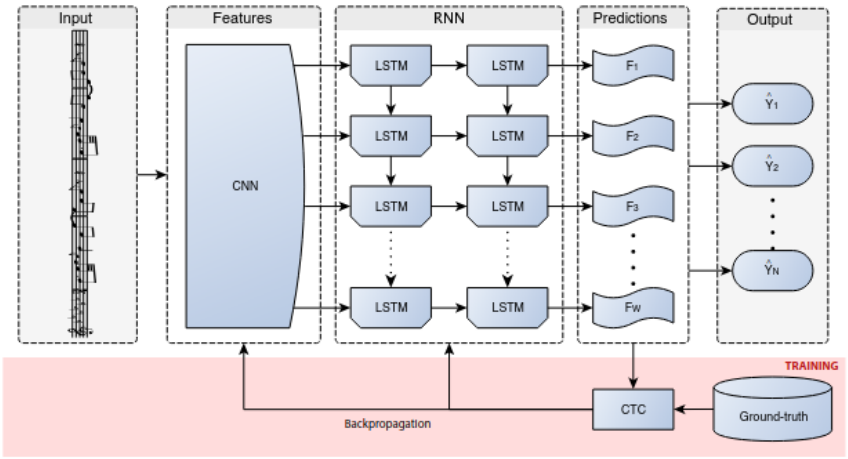
No trabalho de Calvo-Zaragoza e Rizo (2018), foram efetuados os treinos dos modelos através de uma rede neural *end-to-end*, desenvolvida pelos próprios autores. A arquitetura da rede assume que uma seção de pauta monofônica é uma unidade básica, ou seja, as pautas de uma partitura são processadas uma de cada vez. Formalmente, os autores definem o domínio da aplicação com a Equação 2.1, onde x(i) representa uma única imagem de pauta com comprimento variável e y(i) representa a sequência correspondente de símbolos musicais contidos na imagem, cada um dos quais pertence a um conjunto de alfabeto fixo.

$$X = \{(x(i), y(i)), (x(i+1), y(i+1)), (x(i+2), y(i+2)), ...\} \tag{2.1}$$

A rede neural combina as capacidades de *Convolutional Neural Network* (CNN) para trabalhar na imagem de entrada, extraindo as características e evoluindo seu aprendizado, e *Recurrent Neural Network* (RNN) para lidar com a natureza do problema, produzindo a sequência de símbolos musicais que mais corresponde com a imagem. Essa estrutura é baseada em *Connectionist Temporal Classification* (CTC), classificação que permite treinar a rede diretamente a partir das imagens de entrada acompanhadas de suas transcrições correspondentes.

O trabalho de Calvo-Zaragoza e Rizo (2018) foi conduzido sobre um cenário de aprendizagem supervisionado, ou seja, o subconjunto de X com o qual o modelo foi treinado é conhecido. Uma vez que ambos os tipos de redes (CNN e RNN) representam modelos de avanço, a fase de formação pode ser realizada em conjunto, o que leva a uma *Recurrent Neural Network* (CRNN).

Figura 17 – Esquema da abordagem neural *end-to-end*.



Fonte: Adaptado de Calvo-Zaragoza e Rizo (2018)

2.3.3 Reconstrução da notação musical

Esta etapa tem como objetivo reconstruir a semântica musical, atribuindo um significado para o reconhecimento como um todo. Conforme Shatri e Fazekas (2020), sistemas OMR são bidimensionais, isto é, podem ser capazes de reconhecer a sequência e também o relacionamento entre as notas, através da identificação espacial dos objetos reconhecidos. Entretanto, não é uma etapa simples, pois conforme mais relacionamentos de símbolos na pauta, a complexidade da reconstrução também aumenta. Além disso, algumas composições podem não seguir uma regra musical exata de escrita, pois é comum alguns músicos omitirem símbolos para favorecer a legibilidade da composição. O desafio, nesta fase, é modelar uma representação de saída musical que seja semelhante da imagem de entrada, considerando a sua semântica.

Durante a etapa de reconstrução, as regras musicais devem estar sempre sendo verificadas, para que no momento da atribuição do significado não ocorram erros. Fórmulas de compasso provém informações sobre sua duração para detectar a falta ou o excesso de símbolos musicais dentro de um compasso.

Em seu estudo, Ng *et al.* (1999) utilizam o agrupamento da classificação das primitivas para reconstruir o significado semântico. Com isso, é encontrada a tonalidade do símbolo e através de um classificador de vizinhança são buscadas outras características que possam estar relacionadas. Entretanto, o processo desse estudo não é confiável para partituras manuscritas, pois depende de símbolos com bordas retas.

No geral, para determinar a tonalidade de uma nota, é necessário saber seu posicionamento vertical na pauta. Com este parâmetro calculado é possível compará-lo com os comprimentos de referência, obtidos na etapa de estimativa do SLH e SSH. Entretanto, Pacha e Calvo-Zaragoza (2018) utilizam a técnica de CNN como forma alternativa. Neste método, ao detectar uma nota musical, a região da imagem que contém essa nota é extraída e fornecida como entrada para um modelo de CNN, que faz uma predição de qual espaço ou linha a nota está localizada.

Outro método é introduzido por Pacha, Calvo-Zaragoza e Hajic (2019) para determinar os relacionamentos entre símbolos primitivos. Cada símbolo detectado é modelado como sendo um vértice de um grafo, que pode estar conectado com outros vértices através de suas arestas. As arestas podem ser definidas como sintáticas, isto é, que conectam dois símbolos relacionados sintaticamente, como uma cabeça de nota e uma haste; ou podem ser definidas como arestas de precedência, que conectam símbolos relacionados temporalmente, como duas notas em sequência. Esse método busca determinar qual é o tipo de uma aresta que conecta um par de símbolos. Para isso, são utilizadas regras de filtragem de vértices, que se aplicam em símbolos que podem possuir algum relacionamento entre si. Parâmetros de distância e o tipo dos vértices envolvidos também são utilizados. O objetivo dessa filtragem é evitar que todos os pares possíveis de vértices sejam testados.

2.3.4 Codificação da representação final

A última etapa de um sistema OMR consiste na construção da representação final da música. Nessa etapa, o objetivo é converter o reconhecimento para um arquivo musical de saída. Dentre os mais comuns estão: *Musical Instrument Digital Interface* (MIDI) (MIDI, 2022) e MusicXML (MUSICXML, 2022).

O MIDI é um formato de arquivo que possibilita o armazenamento de metadados sonoros de um reconhecimento musical. Dentre os metadados podem existir os efeitos de dinâmica, andamento, entre outros. O MIDI é o formato mais comum utilizado para a reprodução de partituras.

O MusicXML é baseado no formato XML, sendo utilizado para representar a notação musical. Por outro lado, não armazena metadados de som, impossibilitando uma reprodução musical mais detalhada. O formato MusicXML é mais indicado para a criação e edição de partituras digitais, contudo, para decidir qual codificação de saída melhor se adapta ao projeto, é necessário considerar os requisitos do protótipo para a sua aplicação e finalidade.

3 TRABALHOS RELACIONADOS

Este capítulo é destinado ao estudo de alguns trabalhos relacionados ao tema de pesquisa, que envolvem o estudo de um reconhecedor de partituras. O objetivo é expor algumas técnicas utilizadas em projetos anteriores para o reconhecimento musical.

3.1 RECONHECIMENTO COM REDE NEURAL CONVOLUCIONAL

O projeto de Furukawa e Pedrini (2021) foca no desenvolvimento de um método capaz de transformar uma imagem de partitura em um arquivo de saída MIDI. Os autores utilizam conceitos abordados em seções anteriores: binarização, estimativa de referências e detecção de objetos. Utilizou-se o modelo de rede neural treinado, desenvolvido por Calvo-Zaragoza e Rizo (2018), para a detecção de símbolos em partituras digitais.

Na etapa de pré-processamento, foi feito um comparativo entre a binarização de Otsu e os métodos adaptativos de Niblack e Sauvola. Nessa comparação foram utilizadas imagens de partituras manuscritas e digitais, de diferentes tamanhos. A avaliação dos resultados foi realizada empiricamente através de uma análise visual das imagens produzidas.

Conforme análise de Furukawa e Pedrini (2021), o método de Otsu produziu bons resultados em imagens de diferentes tamanhos e com iluminação uniforme. Já os métodos adaptativos tiveram um bom desempenho em imagens com iluminação não uniforme, no entanto, o valor de limiar, utilizado para a separação dos elementos do plano de fundo e da frente da imagem, interfere no resultado. De acordo com os experimentos feitos pelos autores, um valor de limiar muito pequeno acarreta na inserção de ruídos na imagem, já um valor de limiar muito grande, em imagens com iluminação não uniforme, apresentam um resultado insatisfatório. Os métodos de binarização utilizados no projeto são da biblioteca `scikit-image` da linguagem Python (SCIKIT-IMAGE, 2022).

Para a estimativa dos comprimentos de referência, foi necessário, inicialmente, alinhar a imagem da pauta. Furukawa e Pedrini (2021) geraram imagens com rotação em intervalos de 0.25 graus, de -10 até 10 graus, calculando a máxima local de cada projeção horizontal para encontrar o melhor alinhamento da imagem. Foi possível obter bons valores dos comprimentos de referência através do método RLE utilizado antes da binarização da imagem.

Na etapa de detecção dos símbolos musicais foi utilizado o modelo agnóstico, disponibilizado por TF-END-TO-END (2022) no Github. O treino do modelo foi feito utilizando as imagens do *dataset* PRIMUS (citado na Tabela 2) em conjunto com os arquivos de vocabulário semântico, contendo a representação dos símbolos da imagem em forma de texto. Conforme Furukawa e Pedrini (2021), o modelo escolhido é capaz de detectar símbolos em partituras digitais e também em fotos binarizadas, desde que tenham uma qualidade adequada.

As notas são detectadas de forma completa, isto é, não existe uma segmentação por primitivas de notas (cabeça, haste e colchete). Além disso, o modelo não é capaz de detectar notas em partituras que não sejam monofônicas. Como resultado, é produzida uma lista de palavras representando o tipo do símbolo e em qual linha ou espaço da pauta ele se encontra; por exemplo, a `note.quarter-L3`, que indica uma semínima na terceira linha da pauta.

Para a reconstrução da notação musical, Furukawa e Pedrini (2021) aplicaram regras espaciais e musicais para determinar a altura e duração das notas. Entretanto, alguns símbolos não foram classificados corretamente, explicam os autores:

> Como o modelo utilizado para a etapa de detecção de símbolos não produz retângulos envolventes, alguns símbolos são impossíveis de terem seu significado extraído corretamente. Um acidente localizado no começo de uma pauta na maioria das vezes pertence à armadura da clave, mas também é possível que este acidente esteja atrelado à primeira nota da pauta. Em ambos os casos, a saída do modelo agnóstico é a mesma e não é possível distingui-las. Caso fosse utilizado o modelo semântico, seria possível diferenciar essas duas situações.

Considerando o modelo agnóstico escolhido, para garantir um bom reconhecimento, Furukawa e Pedrini (2021) colocaram algumas restrições ao definir como padrão a clave de Sol e o compasso 4/4, por serem configurações de pauta mais simples.

Para a codificação da representação final, os autores optaram pelo formato MIDI, utilizando a biblioteca `MIDIUtil` do Python (MIDIUTIL, 2022). O arquivo final foi testado e reproduzido com sucesso pelos programas Musescore (MUSESCORE, 2022) e Synthesia (SYNTHESIA, 2022). Entretanto, para partituras que não apresentaram regras musicais exatas, a conversão final não ocorreu da forma esperada.

3.2 RECONHECIMENTO COM SVM E ORB

O trabalho de Silva (2017) apresenta uma metodologia para a sistematização de um reconhecedor de partituras manuscritas. O autor utilizou o algoritmo *Oriented FAST and Rotated BRIEF* (ORB) (ORB, 2022) para a extração das características sob um *dataset* de imagens de partituras manuscritas. Para classificação e validação dos símbolos musicais, foram utilizados os métodos de *Brute Force Matcher* (BFM) (JAKUBOVIć; VELAGIć, 2018) (BFM, 2022), *Bag of KeyPoints* (BOK) (CSURKA *et al.*, 2004) e SVM (NOBLE, 2006) (SVM, 2022).

Os resultados atingiram 98,50% de acurácia para a classificação de 20 classes de símbolos, 100% na etapa de detecção de pautas, 98,28% no processo de remoção das pautas, 86,12% no estágio de identificação das posições das notas e 66,12% no processo de reconhecimento das barras de compasso.

Na etapa de pré-processamento, Silva (2017) também utiliza a técnica de binarização pelo método de Otsu, além da operação morfológica de dilatação, com o objetivo de remover

o ruído *"salt and pepper"*. Conforme o autor, o processo de dilatação após a aplicação de um elemento estruturante horizontal, resulta em uma imagem quase sem falhas nas linhas de pauta.

No processamento da pauta, é feita a operação de detecção das linhas. Para isso, Silva (2017) utilizou o algoritmo de Dalitz (DALITZ *et al.*, 2008) em conjunto com a transformada de Hough (ILLINGWORTH; KITTLER, 1988) (HLT, 2022). Conforme autor, esta metodologia aplicada é bastante utilizada para a detecção de linhas, círculos e elipses:

> Os segmentos detectados são empregados na formação de um grafo com caminhos de menor custo. Os segmentos são ordenados por suas posições em altura e largura, então é verificado o caminho estável entre as duas margens utilizando o algoritmo, proposto por Dalitz chamado de *Line Tracking Height*, que verifica se o caminho percorrido é duas vezes menor que a altura da linha de pauta, pesquisa *pixel* a *pixel* com a finalidade de conectar os segmentos detectados que estejam sobre a linha ou mais próximo possível a ela, enquanto que os segmentos não pertencentes aquela linha de pauta, identificados por meio da utilização de variáveis de folgas são removidos.

Na etapa de processamento dos símbolos musicais, Silva (2017) optou por efetuar a remoção das linhas da pauta utilizando operações morfológicas de erosão e dilatação, com base nas referências obtidas anteriormente através do algoritmo LTH. Para a localização e segmentação dos símbolos, foi utilizado o algoritmo de *FloodFill* (BURTSEV; KUZMIN, 1993) (FLOODFILL, 2022), que possibilita encontrar regiões conectadas. Esse algoritmo retorna um vetor contendo todas as regiões conectadas entre si e através da mesma cor.

Para a extração das características, as imagens segmentadas e binarizadas são, então, submetidas ao método ORB, em conjunto ao algoritmo *matcher* BFM (técnica que busca em todos os vetores de características, quais vetores mais se aproximam) e sobre a distância de Hamming (GRANA *et al.*, 2013) (utilizada para denotar a diferença entre duas cadeias binárias). Conforme Silva (2017), por conta do uso do método *holdout* (KOHAVI, 1995), para garantir a generalização dos dados do modelo, a construção do dicionário teve que ser repetida inúmeras vezes, assegurando um baixo desvio sistemático do valor real para os dados.

Nestas condições, a extração resultou em 577 características que foram armazenadas em um arquivo de formato *Extensible Markup Language* (XML) (XML, 2022). Para cada instância de treinamento e teste, foram utilizados conjuntos de dados mutuamente exclusivos, tornando necessária a reconstrução do vocabulário visual, bem como a clusterização dos dados a cada nova etapa de treinamentos e testes. O procedimento é repetido até que, finalmente, o arquivo contendo os histogramas de características para o classificador SVM é retornado.

Para Silva (2017), o algoritmo ORB apresentou-se bastante eficiente, conseguindo extrair características relevantes para a criação do BOK e classificação no SVM. Entretanto, falhou ao encontrar objetos estruturalmente parecidos. Contudo, não inviabiliza o resultado final e permanece como um método relevante para a detecção de símbolos manuscritos.

Na etapa de classificação, foram utilizados os histogramas de características, como vetores de entrada no classificador SVM. No trabalho, a metodologia de particionamento utilizada foi a *holdout*, também conhecida como teste de estimativa de amostras. Os dados foram particionados em grupos: treino: 20% e teste: 80%; treino: 40% e teste: 60%; treino: 60% e teste: 40%; treino: 80% e teste: 20%.

Conforme a metodologia, também foi necessário estimar os valores de *C* e *gamma*. A solução encontrada consiste na utilização da pesquisa em grade (também conhecida como *grid search*) sobre os dois parâmetros citados. Nesse processo, vários testes foram realizados, para que, ao final, os parâmetros de maior precisão fossem escolhidos. Para Silva (2017), a sistematização do reconhecimento de símbolos e sua validação apresentaram resultados confiáveis, além de permitirem uma realimentação do sistema com novas instâncias detectadas, propiciando um ciclo de construção cada vez mais consistente e adaptável.

Para a reconstrução da notação musical, Silva (2017) propôs o uso da transformada de Hough para a detecção de segmentos de reta nas estruturas primitivas das notas musicais. Conforme o autor, é possível reconhecer os elementos primitivos responsáveis pela construção de quase todas as notas musicais (hastes, cabeças e colchetes):

> Portanto, se for efetuado a desagregação das formas primitivas, separando-as dos outros elementos, então torna-se possível a detecção, por exemplo, das cabeças das notas, permitindo assim reconhecer a sua posição em relação a sua linha de pauta. Logo, a remoção dos segmentos de reta, correspondentes às hastes e a aplicação de operações morfológicas de dilatação sobre um elemento estruturante na forma de elipse, torna possível a identificação das posições das notas na pauta.

Por fim, para codificação da representação final, o autor utilizou o formato MusicXML, pois entende que é um formato voltado para o compartilhamento de partituras entre diversas aplicações disponíveis no mercado, de forma robusta e ampla, considerando soluções de código-aberto. Como resultado, o arquivo de retorno apresentou um acerto aproximado a 70% sobre a partitura original, em decorrência das variações na notação musical, resultantes das classificações incorretas das categorias de símbolos extraídos.

3.3 RECONHECIMENTO COM K-NEAREST NEIGHBOUR

O trabalho de Mauricenz (2013) descreve o desenvolvimento de um protótipo em linguagem Java, capaz de reconhecer símbolos musicais a partir de uma imagem de partitura. Para o processo de classificação fez-se necessária a remoção das linhas de pauta através de projeção horizontal, o isolamento dos símbolos musicais através de operações morfológicas (erosão e dilatação) e, também, o cálculo dos descritores de Fourier, caracterizando as formas e utilizando árvore de decisão como método de classificação. Conforme autor, os resultados do protótipo demonstraram uma taxa de 77% de acerto.

Para a binarização da imagem, foi utilizada a função da biblioteca JavaCV (JAVACV, 2022), chamada de `cvThreshold`. Conforme o autor, essa função recebe cinco parâmetros: a imagem de referência, a imagem de destino, o valor do limiar, o novo valor de cor que será aplicado e o tipo do filtro `CV_THRESH_BINARY`, que define o valor do *threshold* usado para a binarização.

Após a etapa de binarização, Mauricenz (2013) subdividiu a etapa seguinte em três passos: remoção das linhas de pauta, aplicação de operações morfológicas e segmentação. Para a remoção das linhas de pauta, foi utilizada a técnica de projeção horizontal, tendo em vista que o protótipo considera imagens em rotação horizontal. Após a projeção horizontal, foi aplicada a remoção das linhas da pauta, atribuindo a cor branca às linhas identificadas. Nesta parte, as linhas da imagem contendo as maiores quantidades de *pixels* pretos são identificadas como as linhas da pauta. O autor admitiu uma variação de até 15% no valor de referência, permitindo uma pequena margem de erro no comprimento das linhas de pautas deformadas.

Conforme já visto na fundamentação teórica, o método utilizado para a remoção das linhas pode causar deformações nos símbolos musicais e, em muitos casos, dividi-los em várias partes. Para corrigir isso, Mauricenz (2013) aplicou um ajuste utilizando as funções de morfologia matemática `cvErode` e `cvDilate`, também do JavaCV. Conforme o autor, esse processo efetua primeiro a operação de erosão, seguida da dilatação, preservando os símbolos. A sequência é respeitada pois, utilizando apenas a dilatação, símbolos muito próximos podem ser fundidos, por outro lado, se essa técnica ocorrer primeiro, as hastes dos símbolos e outras partes finas poderão desaparecer.

Foram realizados testes para avaliar a etapa de remoção das linhas, esperando que, para todas as imagens de entrada, o protótipo consiga identificá-las e removê-las. As imagens fornecidas já haviam passado pelo processo de binarização e, a partir dos experimentos, Mauricenz (2013) conclui que o protótipo identificou e removeu completamente as linhas de pauta, sem restar ruídos.

A etapa de segmentação foi responsável por encontrar os símbolos musicais formados pelos *pixels* pretos em componentes conexos. Para cada forma deveriam ser identificados os contornos, possibilitando o cálculo dos descritores de Fourier, que são utilizados para a classificação das formas.

Para o experimento de segmentação, Mauricenz (2013) selecionou as amostras sem as linhas de pauta, resultantes do processo anterior. O protótipo identificou a forma encontrada e a colocou dentro de uma *bounding box* (caixa de marcação que envolve cada forma) com um valor de margem definido. Contudo, através dos experimentos de segmentação, o autor conclui que o protótipo reconstruiu, em segmentos, os símbolos contidos nos trechos das partituras selecionadas. Entretanto, ocorreu uma exceção nas fórmulas de compasso devido a proximidade dos símbolos, evidenciando que as formas muito próximas também podem ser fundidas na reconstrução, devido a operação morfológica de dilatação.

O autor dividiu a etapa de pós-processamento em três passos: criação da lista de pontos a partir do contorno da forma encontrada, cálculo dos descritores de Fourier e aplicação da classificação entre formas. A etapa de criação da lista de pontos do contorno é responsável por unir os valores de coordenadas úteis para a formação dos descritores de Fourier. Para isso, são geradas duas listas de `pontoX` e `pontoY`.

A etapa do cálculo dos descritores de Fourier é responsável por encontrar os valores de cada descritor, que representam a forma do objeto através do seu contorno. A precisão da forma descrita está diretamente ligada à quantidade de descritores utilizados, isto é, quanto mais descritores, mais precisão e aproximação do resultado à forma original. A quantidade máxima de descritores é a própria quantidade de pontos do contorno. Para isso, são identificados os valores da quantidade de descritores a serem calculados e a quantidade de pontos do contorno para, então, utilizar a transformada discreta de Fourier, com o objetivo de encontrar as características dos objetos.

Figura 18 – Fórmula da transformada discreta de Fourier.

$$a(u) = \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K}$$

Fonte: Adaptado de Mauricenz (2013)

Mauricenz (2013) efetuou experimentos para analisar a etapa de extração de características das formas encontradas após o processo de segmentação. Resumidamente, é extraído o contorno de cada forma, gerando uma lista de coordenadas em um plano cartesiano. Essa lista é o critério para a construção dos descritores de Fourier, obtidos pela transformada discreta de Fourier, que são as justificativas para a comparação entre as diferentes formas. Através desse experimento Mauricenz (2013) conclui que os descritores foram gerados corretamente.

Para a classificação, utilizou-se uma árvore de decisão alimentada pelos dados numéricos obtidos pela extração de características. Suas regras foram construídas sobre a análise dos valores de formas já conhecidas. Para a identificação da forma, os valores encontrados foram submetidos ao método de identificação no qual foram comparados entre si, resultando em um rótulo adicionado à lista de formas do programa.

Para experimentos de classificação, Mauricenz (2013) utilizou como entrada a imagem segmentada e, como saída, o próprio resultado gerado pelo protótipo. A saída agregou os rótulos das formas encontradas e os descritores foram classificados conforme as regras da árvore de decisão, refletindo as assinaturas (descritores de Fourier gerados) das formas utilizadas. O autor conclui que não ocorreram notas parcialmente classificadas e as notas corretamente classificadas não possuíam contornos similares. Das vinte e duas formas utilizadas, 77% foram classificadas apenas com descritores de Fourier.

4 TECNOLOGIAS E FERRAMENTAS

O objetivo deste capítulo é introduzir as ferramentas e *datasets* mais importantes para o desenvolvimento deste trabalho. A solução final foi desenvolvida através das ferramentas descritas nas seções abaixo.

4.1 DATASETS DISPONÍVEIS

Dependendo do sistema OMR e do objetivo de sua aplicação, diferentes *datasets* podem ser utilizados. Este tópico é destinado a projetos de reconhecimento que utilizam modelos de inteligência artificial automatizados. Nos artigos estudados (SHATRI; FAZEKAS, 2020), (REBELO *et al.*, 2012), os autores citam *datasets* contendo diferentes representações de partituras: manuscritas, inteiramente digitalizadas ou apenas com as linhas de pauta digitalizadas. Entre as diferentes notações musicais, as mais citadas são a mensural e a CWMN. A Tabela 2 contém os *datasets* mais comuns estudados.

Tabela 2 – *Datasets* mais comuns para sistemas OMR.

Nome	**Forma de gravação**	**Tamanho**	**Formato**	**Utilização**
HOMUS	Manuscritas	15.200 símbolos	Arquivos texto	Classificação de símbolos (*online* e *offline*)
Universal Music Symbol Collection	Conjunto de tipografias + manuscritas	Próximo a 90.000 símbolos	Imagens	Classificação de símbolos (*offline*)
CVC-MUSCIMA	Manuscritas	1.000 imagens	Imagens	Remoção de linhas da pauta e identificação do escritor
MUSCIMA++	Manuscritas	Mais de 90.000 anotações	Imagens, anotações de compassos, MuNG	Classificação de símbolos, detecção de objetos, reconhecimento *end-to-end* e reconhecimento de compassos
PrIMuS	Conjunto de tipografias	87.678 inícios de melodias	Imagens, MEI, codificação simplificada, codificação agnóstica	Reconhecimento ponta a ponta

Fonte: Adaptado de Repositório (2022)

Calvo-Zaragoza e Oncina (2014) introduzem o *dataset Handwritten Online Musical Symbols* (HOMUS), que contém amostras de 32 tipos de símbolos musicais, obtidos de partituras de 100 músicos diferentes. O *dataset Universal Music Symbol Collection* contém um conjunto de símbolos musicais manuscritos e tipográficos de 79 classes, que podem ser usa-

das para treino de modelos. O *dataset* mais comum para a remoção das linhas da pauta é o CVC-MUSCIMA, que contém imagens de partituras escritas por 50 músicos diferentes. Para a construção desse *dataset*, cada músico foi convidado a transcrever as mesmas 20 páginas de música, utilizando a mesma caneta e o mesmo material de papel. Além disso, esse *dataset* inclui músicas monofônicas e polifônicas. O *dataset* MUSCIMA++ é um derivado do CVC-MUSCIMA e é mais indicado para detecção de símbolos. Ele contém símbolos com notações primitivas, de nível superior e assinaturas de chave ou de tempo.

Entretanto, este trabalho utilizou um modelo treinado a partir do conjunto de imagens pertencentes a um *dataset* destinado a sistemas de reconhecimento *end-to-end*, isto é, que atendem todos os processos de ponta a ponta, incluindo a segmentação e classificação dos símbolos musicais. Esse tipo de aplicação utiliza técnicas de aprendizado de máquina. Foi utilizado o *dataset* PRIMUS (PRIMUS-DATASET, 2018), que também possui a versão Camera-PRIMUS, com imagens distorcidas para simular imperfeições. Esse *dataset* possui partituras nos formatos: PNG, MIDI, *Music Encoding Initiative* (MEI) (MEI, 2022), além das codificações semântica e agnóstica, que representam as partituras em uma sequência de palavras específicas.

4.2 BIBLIOTECAS DE DESENVOLVIMENTO

O protótipo OMR desenvolvido pode ser dividido em dois pilares principais: processamento de imagens e aprendizado de máquina. Com base nestas duas áreas, as bibliotecas utilizadas para o desenvolvimento do sistema foram selecionadas, considerando os critérios a seguir:

a) **Recursos**: para um bom desenvolvimento técnico, uma ferramenta com mais recursos que atendam necessidades específicas, facilita a implementação e torna o projeto mais completo.

b) **Usabilidade**: além dos recursos disponíveis, uma ferramenta de desenvolvimento deve ser intuitiva e de fácil entendimento, acelerando processos e deixando a implementação mais ágil.

c) **Conteúdo**: ferramentas e bibliotecas de desenvolvimento que possuem uma comunidade ativa disponibilizam maior conteúdo e material de suporte, permitindo que problemas ou dúvidas sejam sanadas mais rapidamente.

A biblioteca OpenCV (OPENCV-ORG, 2022) foi desenvolvida pela Intel para resolver problemas do âmbito da visão computacional e é considerada a principal ferramenta *open source* para o processamento de imagem (BRADSKI; KAEHLER, 2008). A biblioteca é escrita em C e C++ e roda sobre os sistemas operacionais Linux, Windows e Mac OS X. Além disso, possui suporte para as linguagens Python, Ruby, Matlab, entre outras.

Conforme Bradski e Kaehler (2008), a OpenCV foi desenvolvida com foco em apli-

cações de tempo real, isto é, aplicações com tarefas concorrentes que se comunicam e se sincronizam. Desta forma é permitido seu uso com processadores *multicore*, além de se tornar compatível para desenvolvimento de código direcionado a *Graphics Processing Unit* (GPU).

A OpenCV fornece uma infra-estrutura simples, o que ajuda na construção rápida de aplicações sofisticadas. A biblioteca contém mais de 500 funções que abrangem desde a área fabril e robótica até a área médica (BRADSKI; KAEHLER, 2008). Além disso, a ferramenta também contém funcionalidades de uso geral para aprendizado de máquina, mas com foco no reconhecimento estatístico de padrões e agrupamento.

A biblioteca TensorFlow (TENSORFLOW-ORG, 2022), que foi desenvolvida pela Google em novembro de 2015, possui código aberto e é voltada para o aprendizado profundo. Essa ferramenta foi utilizada por TF-END-TO-END (2022) para a criação da rede neural convolucional e recorrente, baseada em treinamento e teste CTC. É considerado um *framework* robusto para soluções de alta performance. Conforme Abrahams *et al.* (2016), o principal objetivo dessa biblioteca é fornecer um amplo conjunto de funções e classes que permitam a definição de modelos matemáticos e ofereçam suporte a funcionalidades de aprendizado de máquina.

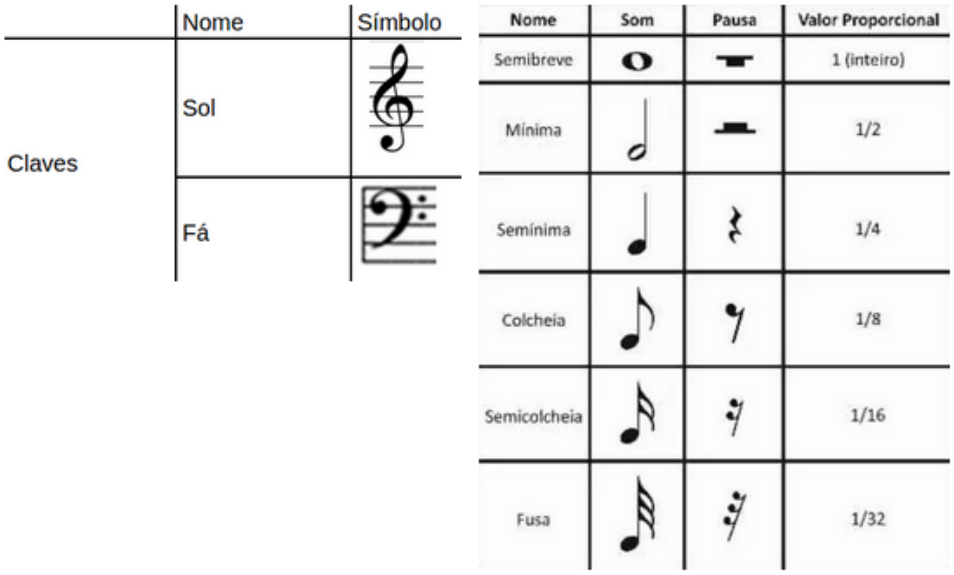
A *Application Programming Interface* (API) da biblioteca TensorFlow permite a definição e o treino de modelos de aprendizado de máquina, cuja camada mais alta é desenvolvida em Python, enquanto seu *core* é escrito em C++. Conforme Abrahams *et al.* (2016), isso foi pensado para disponibilizar uma fácil utilização em conjunto com um código compilado e eficiente.

5 METODOLOGIA

Neste capítulo são descritos os métodos e as abordagens utilizadas no desenvolvimento do reconhecedor de partituras monofônicas, que se encontra no repositório do Github OMR-SISTEMA (2022). Conforme visto em seções anteriores, um sistema de reconhecimento de partituras pode ser utilizado em diferentes aplicações, entretanto, neste projeto foi desenvolvido um sistema de reconhecimento de pautas musicais, avaliando os diferentes métodos de processamento de imagens abordados na fundamentação teórica.

O sistema foi desenvolvido utilizando o modelo treinado de Calvo-Zaragoza e Rizo (2018), o qual permite o reconhecimento de diversos símbolos musicais. Neste trabalho, os símbolos avaliados são apenas uma amostra dentre todas as possibilidades que o modelo utilizado disponibiliza. A Figura 19 demonstra os símbolos avaliados na etapa de classificação e reconstrução musical.

Figura 19 – Claves de Sol e Fá (esquerda). Notas e pausas (direita).

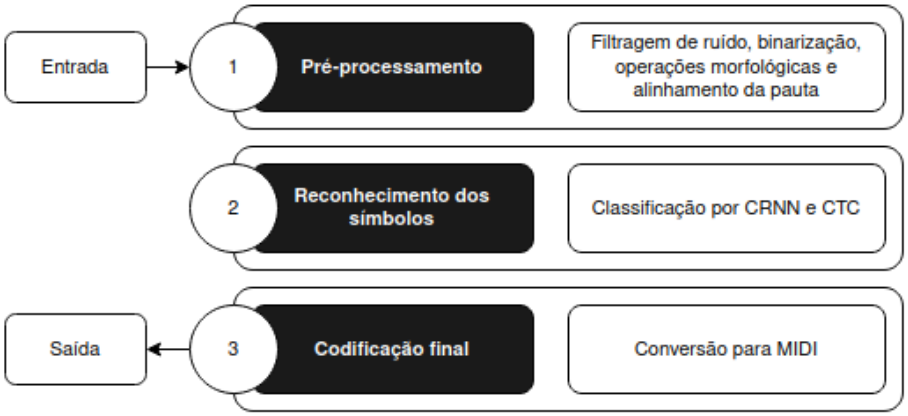


Fonte: O Autor (2022)

5.1 ARQUITETURA DO SISTEMA

A arquitetura utilizada no reconhecedor baseia-se na ideia de subdivisão de tarefas feita por Shatri e Fazekas (2020). No entanto, o sistema desenvolvido foi estruturado em apenas três etapas: pré-processamento de imagem, reconhecimento de símbolos musicais e codificação da representação final. Essa subdivisão possui algumas diferenças nos processos, conforme a Figura 20. O código foi construído através da linguagem Python, que oferece todas as bibliotecas descritas nas seções anteriores (OPENCV, 2022), (TENSORFLOW, 2022), (MIDIUTIL, 2022).

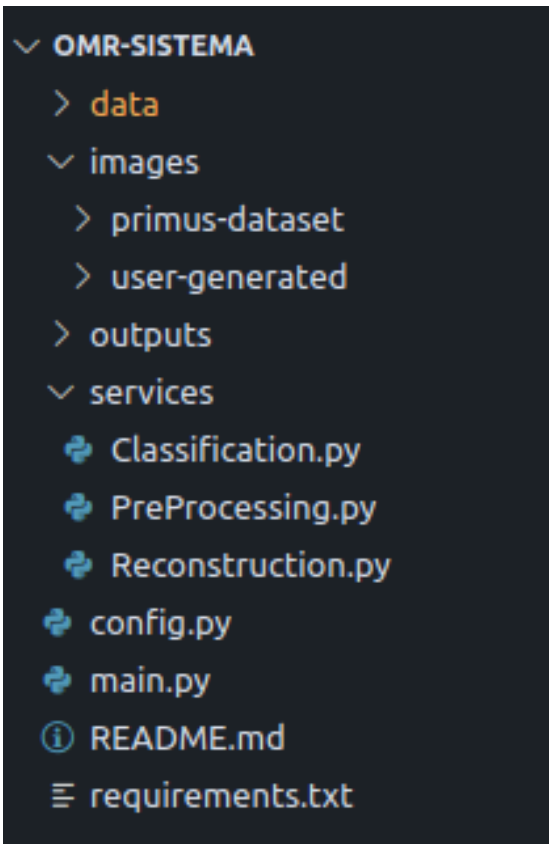
Figura 20 – Arquitetura do sistema OMR.



Fonte: O Autor (2022)

Conforme demonstra a Figura 21, as classes e arquivos do sistema foram organizadas em diferentes diretórios, onde o `main.py` é o ponto inicial da aplicação, responsável por embutir as dependências do sistema (`config.py`) e controlar o fluxo de chamadas dos serviços necessários para o reconhecimento das pautas (`services/`).

Figura 21 – Organização de arquivos do sistema OMR.



Fonte: O Autor (2022)

Algoritmo 1 – Arquivo config.py com as dependências do sistema.

```
1  import warnings
2  warnings.simplefilter(action='ignore', category=FutureWarning)
3
4  import os
5  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
6
7  import sys
8  import json
9  import cv2 as cv
10 import numpy as np
11 import tensorflow as tf
12 from midiutil import MIDIFile
13
14 BASE_DIR = os.path.dirname(os.path.abspath(__file__))
```

Fonte: O Autor (2022)

No diretório `services/` foram desenvolvidos os serviços principais do reconhecedor, seguindo o diagrama da Figura 20. O diretório `data/` armazena o modelo semântico e seu vocabulário completo, importados do repositório TF-END-TO-END (2022).

Figura 22 – Parte do arquivo de texto com o vocabulário de palavras.

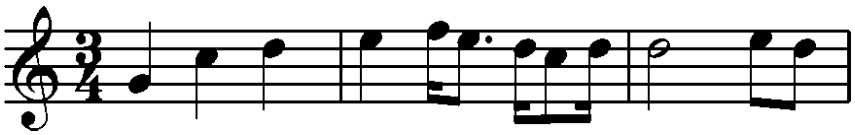
```
589  note-B#4_half
590  note-B#4_half.
591  note-B4_half
592  note-B4_half.
593  note-B4_half_fermata
594  note-B4_half._fermata
595  note-B4_quadruple_whole
596  note-B#4_quarter
597  note-B#4_quarter.
598  note-B4_quarter
599  note-B4_quarter.
600  note-B4_quarter..
```

Fonte: O Autor (2022)

O arquivo de vocabulário possui todas as representações dos símbolos musicais em formato de texto, separando as palavra por quebra de linha. Cada palavra possui uma determinada estrutura, seguindo o tipo de símbolo com o qual está relacionada. No caso das figuras de notas, a estrutura segue a seguinte regra: tipo do símbolo - nota musical - tonalidade - tempo da nota; no caso das pausas: tipo da pausa - tempo da pausa. Essas estruturas compõem a representação dos respectivos exemplos: `note-G5_quarter` (Semínima de Sol na tonalidade cinco) e `rest-eighth` (Colcheia de pausa).

No diretório `images/` ficam as imagens extraídas do *dataset* PRIMUS e as imagens geradas manualmente através do *software* Noteflight (NOTEFLIGHT, 2022). Foram selecionadas e geradas 14 imagens para cada *dataset*, com o objetivo de validar a solução e avaliar os resultados sobre diferentes parâmetros de entrada.

Figura 23 – Exemplo de imagem do *dataset* PRIMUS.



Fonte: Adaptado de PRIMUS-DATASET (2018)

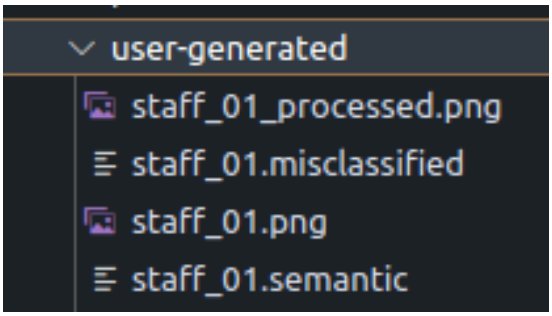
Figura 24 – Exemplo de imagem gerada no *software* Noteflight.

Fonte: O Autor (2022)

As imagens do *dataset* PRIMUS estavam totalmente alinhadas, já as imagens do *dataset* próprio foram geradas com menor qualidade e foram giradas em seu eixo central para fins de validação dos métodos de processamento e de alinhamento das pautas, utilizando a projeção horizontal.

Além das imagens originais e pré-processadas, o diretório `images/` armazena as representações textuais (`staff_01.semantic`), utilizadas para a validação dos resultados de classificação, e um arquivo contendo os erros de classificação (`staff_01.misclassified`) de cada pauta, conforme a Figura 25.

Figura 25 – Diretório de imagens do sistema OMR.



Fonte: O Autor (2022)

O diretório `outputs/` armazena os arquivos `.mid` que são gerados após a classificação e reconstrução da representação musical de cada pauta reconhecida, entretanto, a codificação para o formato MIDI só acontece caso o acerto da classificação seja de 100%, garantindo, desta forma, a geração do arquivo musical sem erros.

5.1.1 Pré-processamento de imagem

Para o pré-processamento foi desenvolvida a classe `PreProcessing`. Conforme visto em seções anteriores, a OpenCV é escrita em C e C++, o que a torna bastante eficiente. Através desta biblioteca, foram desenvolvidos métodos para a filtragem de ruídos, binarização e alinhamento de pauta. Dentre os métodos está o filtro Gaussiano (Algoritmo 2), utilizado para a remoção de ruídos e efeito de suavização das linhas dos objetos da imagem. Contudo, dependendo do tamanho da máscara utilizada, o filtro Gaussiano pode causar um efeito de borrão.

Algoritmo 2 – Método para aplicação do filtro Gaussiano.

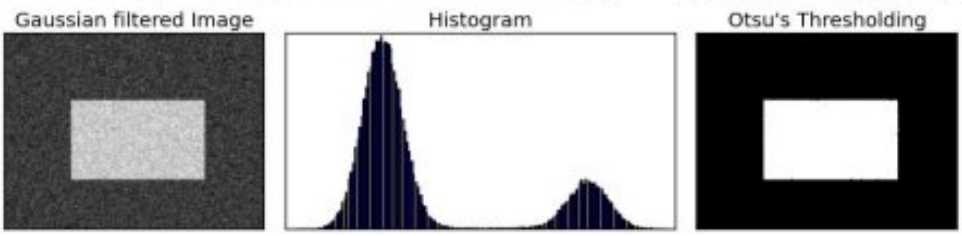
```
1 def __apply_gauss_filter(self, img, mask = (3, 3)):
2         img_filt = cv.GaussianBlur(img, mask, 0)
3         return img_filt
```

Fonte: O Autor (2022)

Por padrão, o método do filtro Gaussiano utiliza um *kernel* de tamanho 3x3; entretanto, também foram feitas simulações com os tamanhos 5x5 e 7x7. A função `cv.GaussianBlur` recebe o valor de `SigmaX`, que permite definir o desvio padrão na direção x. Quando o valor do `SigmaX` é 0, o cálculo é feito em função do tamanho do *kernel*.

Conforme exemplo da Figura 26, a utilização do filtro Gaussiano antes da binarização de Otsu resulta em uma filtragem de melhor qualidade, pois o histograma da imagem filtrada inicialmente possui picos mais distintos, e a binarização de Otsu funciona especialmente bem para casos de imagens com histograma bimodal (dois picos existentes).

Figura 26 – Resultado de aplicação dos filtros Gaussiano e de Otsu, respectivamente.



Fonte: Adaptado de OpenCV (2022)

Além do método Gaussiano foram desenvolvidos os filtros de binarização Global e de binarização de Otsu, que obteve bons resultados nos estudos feitos (FURUKAWA; PEDRINI, 2021), (OTSU, 1979).

Algoritmo 3 – Método para binarização Global.

```
1 def __apply_global_binarization(self, img):
2         ret, img_bin = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
3         return img_bin
```

Fonte: O Autor (2022)

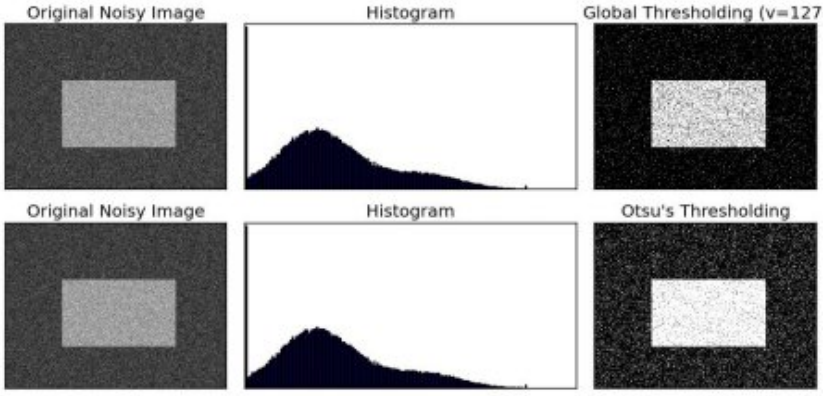
Algoritmo 4 – Método para binarização de Otsu.

```
1 def __apply_otsu_binarization(self, img):
2         thresholdingType = cv.THRESH_BINARY + cv.THRESH_OTSU
3         ret, img_bin = cv.threshold(img, 0, 255, thresholdingType)
4         return img_bin
```

Fonte: O Autor (2022)

No Algoritmo 3 (Global) foi utilizado um valor arbitrário escolhido como limiar; isto é, *pixels* com código de cor maior ou igual a 127 serão brancos e com código de cor menor serão pretos. Em contraste, o Algoritmo 4 (Otsu) determina o valor ideal do threshold utilizado para a separação dos elementos do fundo e da frente da imagem.

Figura 27 – Comparação da aplicação da binarização Global de Otsu. Imagem original (esquerda), histograma original (centro) e resultado da binarização (direita).



Fonte: Adaptado de OpenCV (2022)

Além da binarização, um método para processamento adicional com operações morfológicas de erosão e dilatação também foi desenvolvido, conforme Algoritmo 5.

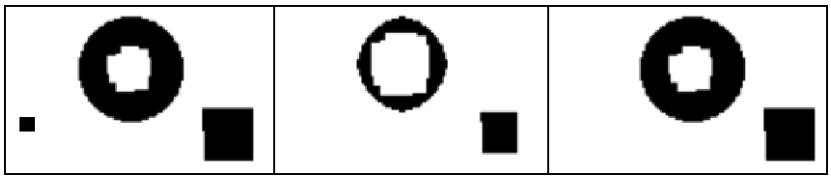
Algoritmo 5 – Método para aplicação de operações morfológicas.

```
1 def __apply_morphological_operations(self, img, mask = (3, 3)):
2         img_transformed = img
3         kernel = np.ones(mask, np.uint8)
4         img_transformed = cv.erode(img_transformed, kernel, iterations=1)
5         img_transformed = cv.dilate(img_transformed, kernel, iterations=1)
6         return img_transformed
```

Fonte: O Autor (2022)

A erosão diminui os elementos da imagem e pode eliminá-los caso tenham tamanho inferior ao *kernel* aplicado, aumentando buracos e até mesmo separando objetos da imagem. Já a dilatação resulta em um efeito contrário, aumentando os elementos, preenchendo os buracos e conectando os objetos mais próximos. O uso em conjunto pode remover ruídos que sejam menores que o *kernel* utilizado, conforme exemplo da Figura 28.

Figura 28 – Exemplo do uso da erosão seguida da dilatação. Imagem original (esquerda), aplicação da erosão (centro) e aplicação da dilatação (direita).



Fonte: O Autor (2022)

Considerando a prática utilizada por Furukawa e Pedrini (2021), para o alinhamento da pauta foi desenvolvido um método que efetua pequenas rotações na imagem, seguindo os parâmetros definidos: `DEGREES_LIMIT` e `DEGREES_STEP`. A cada passo de rotação os filtros foram aplicados e a projeção horizontal foi calculada, considerando a binarização. Com a imagem filtrada a aplicação das projeções horizontais acontece com maior confiabilidade. Foi possível determinar o melhor valor de máxima local para cada pauta.

Algoritmo 6 – Parte do método para alinhamento da pauta.

```
1  def align_staff(self, bin_method, blur_method, morphological_methods):
2    best_image = self.img
3    maximum_projection = 0
4    rotation_scale = 1
5
6    degrees = np.arange(-self.DEGREES_LIMIT, self.DEGREES_LIMIT,
7                        self.DEGREES_STEP)
8    for degree in degrees:
9      M = cv.getRotationMatrix2D(self.img_sizes_center, degree,
10                                rotation_scale)
11     img_transformed = cv.warpAffine(self.img, M, self.img_sizes[::-1],
12                                     borderValue=(255, 255, 255))
13     ...
```

Fonte: O Autor (2022)

O método `align_staff` recebe por parâmetro os filtros que devem ser aplicados antes de efetuar o alinhamento da pauta. Na linha 6 é gerado o *array* para a iteração dos graus de rotação, com o objetivo de encontrar o melhor valor de máxima local. Nas linhas 9 e 11 são utilizados os métodos da biblioteca OpenCV `cv.getRotationMatrix2D` e `cv.warpAffine`. A operação `cv.warpAffine` permitiu a rotação das imagens para o grau determinado. Foi necessário informar a matriz de rotação como parâmetro, obtida através da função `cv.getRotationMatrix2D`. Para obter a matriz de rotação, as referências de centro da imagem foram utilizadas. Essas referências são calculadas na inicialização da classe e ficam definidas nos atributos `self.img_sizes` e `self.img_sizes_center`, conforme exemplo no Algoritmo 7. Para isso, na construção do objeto é passado o caminho do arquivo para abertura da imagem através do método `cv.imread`.

Algoritmo 7 – Método de inicialização da classe de pré-processamento de imagem.

```
1  class PreProcessing:
2      ...
3      def __init__(self, image_path):
4          self.img_path = image_path
5          self.img = cv.imread(self.img_path, 0)
6          self.img_sizes = self.img.shape[:2]
7          self.img_sizes_center = (self.img_sizes[1] // 2,
8                                   self.img_sizes[0] // 2)
9          self.img_processed_path = None
```

Fonte: O Autor (2022)

O atributo `self.img_sizes` armazena os valores dos comprimentos dos eixos x e y da imagem, já o atributo `self.img_sizes_center` armazena os valores de centro de y e x respectivamente, utilizados nos métodos de rotação citados no Algoritmo 6, onde a variável `rotation_scale` é setada como 1 para gerar uma matriz `M` com a mesma escala da imagem original e a variável `borderValue` é definida com o valor [255, 255, 255] para que o fundo da imagem rotacionada seja branco.

Conforme o Algoritmo 8, a aplicação dos filtros é feita obedecendo os parâmetros enviados para a função `align_staff`. Se as condicionais das linhas 1, 4, 6 ou 9 forem verdadeiras, os métodos de filtragem serão chamados, recebendo por parâmetro as imagens e retornando-as na mesma estrutura de matriz recebida, porém com seus valores alterados.

Algoritmo 8 – Parte do método de alinhamento da pauta e pré-processamento da imagem.

```
1  if blur_method == self.BLUR_METHOD_GAUSS:
2      img_transformed = self.__apply_gauss_filter(img_transformed)
3
4  if bin_method == self.BIN_METHOD_GLOBAL:
5      img_transformed = self.__apply_global_binarization(img_transformed)
6  elif bin_method == self.BIN_METHOD_OTSU:
7          img_transformed = self.__apply_otsu_binarization(img_transformed)
8
9  if morphological_methods == self.MORPH_METHODS_EROSION_DILATION:
10     img_transformed = self.__apply_morphological_operations(img_transformed)
```

Fonte: O Autor (2022)

Para o cálculo das projeções horizontais, foi utilizada a função `cv.reduce`, permitindo a redução da matriz da imagem para um vetor. O parâmetro utilizado foi o de redução de soma (`cv.REDUCE_SUM`), possibilitando obter a maior soma (máxima local) através da função `max` do Python. Conforme exemplo do Algoritmo 9, a variável `mask` armazena a matriz da imagem em formato binário, onde 0 equivale a cor branca e 1 equivale a cor preta, e a variável `dimension_index` indica que a redução de soma da matriz será horizontal, isto é, por coluna.

Algoritmo 9 – Exemplo de código em Python para projeção horizontal.

```
def __get_horizontal_projection(self, img):
    mask = np.uint8(np.where(img == 0, 1, 0))
    dimension_index = 1
    h_projection = cv.reduce(mask,
                             dimension_index,
                             cv.REDUCE_SUM,
                             dtype=cv.CV_32SC1)
    return max(h_projection)
```

Fonte: O Autor (2022)

Após a filtragem e definição da imagem de pauta mais alinhada, foi executado o método desenvolvido para o salvamento do arquivo, conforme demonstra o Algoritmo 10. O arquivo é gravado em memória para ser lido e processado na sequência pela etapa de classificação.

Algoritmo 10 – Exemplo de código em Python para salvamento de imagem pré-processada.

```
def save_img_processed(self, img):
    index = self.img_path.find('.png')
    if not index:
        index = self.img_path.find('.jpg')
    if not index:
        index = self.img_path.find('.jpeg')
    self.img_processed_path = (self.img_path[:index] + '_processed'
                               + self.img_path[index:])
    cv.imwrite(self.img_processed_path, img)
```

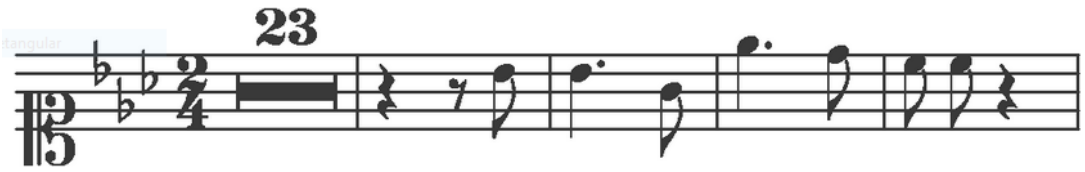
Fonte: O Autor (2022)

5.1.2 Reconhecimento de símbolos musicais

Para a classificação dos elementos foi desenvolvida a classe `Classification`. Dentre os modelos neurais disponíveis, foi utilizado o modelo semântico, que possui um vocabulário de símbolos com significado musical já atribuído, simplificando a fórmula para a reconstrução da notação musical. Esse modelo foi treinado com base no conjunto de dados PRIMUS, o qual possui diversas pautas monofônicas com diferentes configurações. Entretanto, conforme citado no início deste capítulo, as pautas selecionadas para uso neste trabalho possuem apenas os símbolos básicos, como as claves de Sol e Fá, todas as pausas e todas as notas.

É possível perceber a diferença dos vocabulários disponíveis (semântico e agnóstico) no exemplo da Figura 29. A nota Bb4, posicionada no terceiro compasso da pauta, é considerada pelo modelo semântico (Figura 29b) junto ao ponto de aumento, sendo traduzida em apenas uma palavra (`note-Bb4_quarter.`). Já o modelo agnóstico (Figura 29c) a considera de forma diferente, atribuindo a cada símbolo uma palavra junto a sua respectiva posição na pauta (`note.quarter-L4` e `dot-S4`).

Figura 29 – Figura de pauta (a), seu modelo semântico (b) e seu modelo agnóstico (c).



(a)

```
clef-C1 keySignature-EbM timeSignature-2/4 multirest-23 barline rest-quarter rest-eighth note-
Bb4_eighth barline note-Bb4_quarter. note-G4_eighth barline note-Eb5_quarter. note-D5_eighth barline
note-C5_eighth note-C5_eighth rest-quarter barline
```

(b)

```
clef.C-L1 accidental.flat-L4 accidental.flat-L2 accidental.flat-S3 digit.2-L4 digit.4-L2 digit.2-S5
digit.3-S5 multirest-L3 barline-L1 rest.quarter-L3 rest.eighth-L3 note.eighth-L4 barline-L1
note.quarter-L4 dot-S4 note.eighth-L3 barline-L1 note.quarter-S5 dot-S5 note.eighth-L5 barline-L1
note.eighth-S4 note.eighth-S4 rest.quarter-L3 barline-L1
```

(c)

Fonte: Adaptado de TF-END-TO-END (2022)

Para a predição dos elementos musicais foi desenvolvido o método baseado no código disponibilizado no repositório TF-END-TO-END (2022). Essa referência utiliza a biblioteca TensorFlow, que obtém os parâmetros treinados do modelo para efetuar a segmentação e predição de cada símbolo da pauta, conforme demonstra o Algoritmo 11.

Algoritmo 11 – Método para predição dos elementos musicais.

```
1  def detect_symbols(self):
2      graph = self.__get_model_default_graph()
3      model_input = graph.get_tensor_by_name("model_input:0")
4      seq_lengths = graph.get_tensor_by_name("seq_lengths:0")
5      keep_prob = graph.get_tensor_by_name("keep_prob:0")
6      input_height = graph.get_tensor_by_name("input_height:0")
7      width_reduction = graph.get_tensor_by_name("width_reduction:0")
8      logits = tf.get_collection("logits")[0]
9
10     WIDTH_REDUCTION, HEIGHT = self.sess.run([width_reduction, input_height])
11     decoded, _ = tf.nn.ctc_greedy_decoder(logits, seq_lengths)
12     img = self.__normalize_image(self.img, HEIGHT)
13     seq_len = [img.shape[2] / WIDTH_REDUCTION]
14
15     prediction = self.sess.run(decoded, feed_dict={model_input: img,
16                                seq_lengths: seq_len, keep_prob: 1.0})
17     str_predictions = self.__sparse_tensor_to_strs(prediction)
18
19     classified_symbols = []
20     for w in str_predictions[0]:
21         classified_symbols.append(self.vocabulary_semantic[w])
22     self.classified_symbols = classified_symbols
```

Fonte: O Autor (2022)

Conforme o Algoritmo 12, o método `__get_model_default_graph` é responsável por iniciar a sessão iterativa do TensorFlow, carregar o modelo e restartar os pesos do gráfico, retornando os parâmetros do modelo. Os valores de `WIDTH_REDUCTION` e `HEIGHT` são obtidos para normalizar a imagem e segmentar o símbolo da sequência. Após isso, o método `run` é executado para efetuar a predição.

Conforme a documentação da ferramenta TensorFlow (2022), sessões interativas são utilizadas em contextos interativos, como um Shell (linha de comando usada por usuários em um console virtual ou em um emulador de terminal). O método `run` é acionado para chamar operações ou calcular o valor de um tensor sobre um determinado conjunto (`fetches`), substituindo os valores em `feed_dict` pelos valores de entrada correspondentes. O argumento `fetches` pode ser um único elemento, lista, tupla ou objeto contendo elementos de gráfico em suas folhas. O valor retornado por `run` tem a mesma forma do argumento `fetches`, em que as folhas são substituídas pelos valores correspondentes e retornados pelo TensorFlow.

Algoritmo 12 – Método para inicialização da sessão iterativa e do modelo treinado.

```
1 def __get_model_default_graph(self):
2     tf.reset_default_graph()
3     self.sess = tf.InteractiveSession()
4     # Restaura os pesos
5     model_path = self.__get_model_path()
6     saver = tf.train.import_meta_graph(model_path)
7     saver.restore(self.sess, model_path[:-5])
8     return tf.get_default_graph()
```

Fonte: O Autor (2022)

A predição dos elementos resulta em uma lista numérica que representa a sequência dos símbolos musicais reconhecidos. Esta lista é iterada para obter o significado textual de cada elemento reconhecido, armazenados no atributo `self.classified_symbols`. Para obtenção da forma textual da sequência reconhecida, foi utilizado o vocabulário semântico disponibilizado no repositório TF-END-TO-END (2022), que é inicializado no construtor da classe através do método `__get_vocabulary_semantic`, conforme o Algoritmo 13.

Algoritmo 13 – Método para leitura do arquivo do vocabulário semântico.

```
1 def __get_vocabulary_semantic(self):
2     vocabulary_semantic_file = open(self.__get_vocabulary_path(), 'r')
3     word_list = dict()
4     for word in vocabulary_semantic_file.read().splitlines():
5         word_index = len(word_list)
6         word_list[word_index] = word
7     vocabulary_semantic_file.close()
8     return word_list
```

Fonte: O Autor (2022)

Para a avaliação do classificador, foi desenvolvido o método `calculate_accuracy`, conforme Algoritmo 14. Este método utiliza a representação textual da pauta para validar com a sequência textual reconhecida pelo sistema, somando os acertos. O percentual de acerto final é calculado pela Equação 5.1.

$$percentual\_acerto = total\_acerto/total\_esperado \tag{5.1}$$

Algoritmo 14 – Método para cálculo de acerto do classificador.

```
def calculate_accuracy(self):
    img_aux = self.img_path.replace('_processed.png', '.semantic')
    with open(img_aux, 'r') as f:
        expected_vocabulary = f.readlines()
    vocabulary_length = len(expected_vocabulary)
    total_hits = 0
    i_expected = 0
    i_classified = 0
    misclassified_symbols = []
    while i_expected < vocabulary_length:
        i_expected += 1
        try:
            expected_word = expected_vocabulary[i_expected-1]
            expected_word = expected_word.rstrip()
        except IndexError:
            expected_word = ''
        while i_classified < vocabulary_length:
            i_classified += 1
            try:
                classified_word = self.classified_symbols[i_classified-1]
                classified_word = classified_word.rstrip()
            except IndexError:
                classified_word = ''

            if classified_word == expected_word:
                total_hits += 1
            else:
                misclassified_symbols.append({
                    'position': i_expected,
                    'expected_word': expected_word,
                    'classified_word': classified_word
                })
                break
    if len(misclassified_symbols) > 0:
        self.__save_misclassified_file(misclassified_symbols)
    return {'total': total_hits, 'expected': vocabulary_length}
```

Fonte: O Autor (2022)

Caso a palavra classificada seja a mesma da palavra esperada na sequência, a variável `total_hits` é incrementada em 1, caso contrário, as informações de posição, palavra esperada e palavra classificada são gravadas no atributo `self.misclassified_symbols` e um arquivo texto é gerado no final da execução para fins de análise.

Algoritmo 15 – Exemplo de arquivo de erros de classificação.

```
{"position": 5,
 "expected_word": "note-B4_quarter",
 "classified_word": "note-B4_eighth"}
{"position": 6,
 "expected_word": "note-G4_quarter",
 "classified_word": "note-G4_eighth"}
```

Fonte: O Autor (2022)

5.1.3 Codificação da representação final

Na etapa de codificação da representação final, um arquivo MIDI é gerado caso o percentual de acerto seja de 100%. Desta forma é garantido que o arquivo musical não tenha erros ou falhas relacionadas à semântica da música.

Para a solução, foi desenvolvida a classe `Reconstruction`, na qual foi embutida a biblioteca `midiutil` como dependência, permitindo adicionar os símbolos reconhecidos à faixa musical, de forma sequencial. Além disso, foi possível configurar parâmetros de nome, tempo e assinatura da faixa. O atributo `self.symbols` recebe a lista das palavras que representam os símbolos musicais.

Algoritmo 16 – Inicializador da classe Reconstruction.

```
class Reconstruction:

    def __init__(self, image_path, symbols):
        self.midi_file = MIDIFile(1)
        self.img_path = image_path
        self.symbols = symbols
        self.volume = 100
```

Fonte: O Autor (2022)

Outra dependência adicionada foi a classe `Score`, responsável por tratar a codificação das palavras para o formato que a biblioteca `midiutil` deve receber. O mapeamento é feito em constantes de classe que armazenam o valor da palavra reconhecida e o valor aceito pelo objeto `midiutil`. As constantes são: `CLEFS`, `TIME_SIGNATURES`, `PITCHS`, `OCTAVES`, `DURATIONS` e `DOTS` e foram desenvolvidas considerando os elementos musicais definidos para o reconhecimento neste projeto. A codificação feita do mapeamento das palavras é arma-

zenada no atributo `self.melody`, o qual é iterado posteriormente para gerar o objeto da faixa MIDI final. Esse procedimento foi executado sem erros.

Algoritmo 17 – Classe Score e suas constantes.

```
class Score:

    CLEFS = {
        'clef-G2': 'treble',
        'clef-F4': 'bass'
    }
    TIME_SIGNATURES = {
        'timeSignature-2/4': [2, 4],
        'timeSignature-3/2': [3, 2],
        'timeSignature-3/4': [3, 4],
        'timeSignature-4/4': [4, 4],
        'timeSignature-6/4': [6, 4],
        'timeSignature-6/8': [6, 8],
        'timeSignature-C': [4, 4]
    }
    PITCHS = {
        'C': [24, 36, 48, 60, 72],
        'D': [26, 38, 50, 62, 74],
        'E': [28, 40, 52, 64, 76],
        'F': [29, 41, 53, 65, 77],
        'G': [31, 43, 55, 67, 79],
        'A': [33, 45, 57, 69, 81],
        'B': [35, 47, 59, 71, 83]
    }
    OCTAVES = [1, 2, 3, 4, 5]
    DURATIONS = {
        'whole': 1,
        'half': 2,
        'quarter': 4,
        'eighth': 8,
        'sixteenth': 16,
        'thirty_second': 32,
        'sixty_fourth': 64
    }
    DOTS = {
        '.': 1/2
    }
```

Fonte: O Autor (2022)

Para a população dos atributos da musica `.mid`, foram desenvolvidos métodos que obtém os valores das constantes citadas anteriormente. No caso das claves e da assinatura de tempo, os valores são atribuídos às variáveis `self.clef` e `self.time_signature`, res-

pectivamente. Já para as notas e pausas, os valores são adicionados ao atributo `self.melody`. Basicamente, esses métodos recebem a palavra reconhecida, tratam o mapeamento e adicionam o valor codificado ao atributo da classe referente.

Algoritmo 18 – Métodos da classe Score para adição da melodia.

```
def add_clef(self, symbol):
    self.clef = self.CLEFS[symbol]

def add_time_signature(self, symbol):
    self.time_signature = self.TIME_SIGNATURES[symbol]

def add_note(self, symbol):
    note_params = self.__get_note_params(symbol)

    self.melody.append({
        'note': note_params
    })

def add_rest(self, symbol):
    rest_params = self.__get_rest_params(symbol)

    self.melody.append({
        'rest': rest_params
    })
```

Fonte: O Autor (2022)

No caso das notas e das pausas, foram desenvolvidos dois métodos *getters* adicionais com regras específicas. Para as notas (Algoritmo 20) foram tratadas as questões da tonalidade (`self.PITCHS`), oitava (`self.OCTAVES`), duração (`self.DURATIONS`) e do ponto de aumento (`self.DOTS`), que adiciona mais duração ao símbolo, caso exista. Para as pausas (Algoritmo 19) o tratamento foi desenvolvido apenas para a duração.

Algoritmo 19 – Get de pausa da classe Score.

```
def __get_rest_params(self, symbol):
    duration = None

    for d in self.DURATIONS:
        if d in symbol:
            duration = d
            break

    return {
        'duration': self.DURATIONS[duration]
    }
```

Fonte: O Autor (2022)

Algoritmo 20 – Get de nota da classe Score.

```
def __get_note_params(self, symbol):
    pitch = None
    octave = None
    duration = None
    increase = None

    for p in self.PITCHS:
        if p in symbol:
            pitch = p
            break

    for o in self.OCTAVES:
        if str(o) in symbol:
            octave = o
            break

    for d in self.DURATIONS:
        if d in symbol:
            duration = d
            break

    for inc in self.DOTS:
        if inc in symbol:
            increase = inc
            break

    pitch = self.PITCHS[pitch][octave-1]
    duration = self.DURATIONS[duration] + int(
        self.DURATIONS[duration] * (self.DOTS[inc] if increase else 0))

    return {
        'pitch': pitch,
        'duration': duration
    }
```

Fonte: O Autor (2022)

Por fim, o atributo `score.melody` é iterado para adicionar, de forma sequencial, os valores ao atributo `self.midi_file` (objeto `midiutil`) e gerar o arquivo musical final. Conforme Algoritmo 21, caso exista um elemento de nota, o mesmo é adicionado ao objeto `midiutil` e o seu tempo de duração é acrescido ao tempo total da faixa, caso seja uma pausa, apenas o tempo de duração é acrescido à faixa MIDI.

Como resultado, foram geradas faixas compatíveis com as imagens de pautas reconhecidas e foram testadas e aprovadas com sucesso. O teste foi feito de forma analógica através do entendimento auditivo e leitura da música.

Algoritmo 21 – Etapa de adição das notas e geração do arquivo MIDI.

```
for sound in score.melody:
    duration = 0

    if 'note' in sound:
        note = sound['note']
        self.midi_file.addNote(
            track,
            channel,
            note['pitch'],
            time,
            score.time_signature[1],
            self.volume
        )
        duration = note['duration']
    elif 'rest' in sound:
        rest = sound['rest']
        duration = rest['duration']

    time += score.time_signature[1] / duration

self.__save_midi_file()
```

Fonte: O Autor (2022)

Algoritmo 22 – Método para geração do arquivo MIDI.

```
def __save_midi_file(self):
    midi_name = self.img_path.replace('_processed.png', '.mid')
    midi_name = midi_name.split('/')[-1]
    midi_path = 'outputs/' + midi_name

    with open(midi_path, 'wb') as output_file:
        self.midi_file.writeFile(output_file)
```

Fonte: O Autor (2022)

6 RESULTADOS E ANÁLISES

Este capítulo apresenta os resultados obtidos com a execução do sistema de reconhecimento, comparando com diferentes parâmetros de entrada e configurações de pré-processamento, abordando análises da precisão de acertos e de erros de classificação.

6.1 PRECISÃO DE ACERTO

O sistema foi executado com diferentes configurações de filtragem de imagem. Dentre os filtros disponíveis, o único obrigatório é o de binarização, que deve ser sempre aplicado considerando as opções disponíveis: binarização Global ou de Otsu. O restante dos filtros não são obrigatórios e podem ser ignorados no momento de execução.

Tabela 3 – Parâmetros de filtragem disponíveis no sistema.

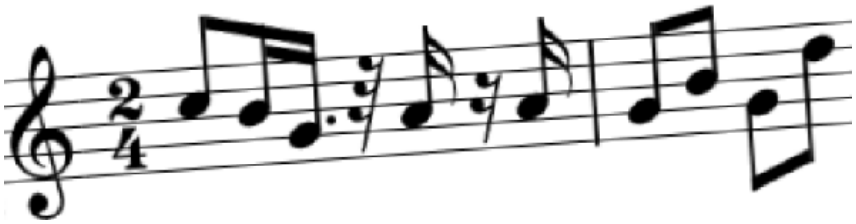
Tipo	**Nome**	**Valor**
Binarização	BIN_METHOD_GLOBAL	1
Binarização	BIN_METHOD_OTSU	2
Filtragem de ruído	BLUR_METHOD_NONE	0
Filtragem de ruído	BLUR_METHOD_GAUSS	1
Operação morfológica	MORPH_METHODS_NONE	0
Operação morfológica	MORPH_METHODS_EROSION_DILATION	1

Fonte: O Autor (2022)

O sistema OMR foi executado tanto para o grupo de imagens selecionadas do *dataset* PRIMUS quanto para o grupo de imagens geradas através do *software* Noteflight. O reconhecedor foi adaptado para suportar imagens de entrada em lote, e conforme citado anteriormente, considera cada imagem de pauta como um vetor de n símbolos, que são classificados e avaliados individualmente de forma iterativa. O sistema OMR soma os acertos dos símbolos classificados e soma o total de símbolos esperados (presentes no arquivo de vocabulário de cada pauta), repetindo o processo para cada imagem de pauta iterada. Após ter obtido a soma total de todos os símbolos das pautas, o percentual de classificação foi calculado utilizando a Equação 5.1.

Os dois grupos de *datasets* selecionados possuíam um total de 14 imagens, resultando em 14 execuções para cada grupo. As imagens do *dataset* PRIMUS totalizavam em 311 símbolos e as imagens do *dataset* próprio totalizavam em 252 símbolos musicais, considerando as barras que separam cada compasso, pois também são interpretadas pelo classificador. As imagens do *dataset* PRIMUS obtiveram os melhores resultados e com menor variação, no entanto, para a demostração dos resultados foram utilizadas as imagens do *dataset* próprio. Vale ressaltar que apenas os resultados de classificação e dos arquivos MIDI foram avaliados neste projeto, excluindo os parâmetros de performance e tempo de execução.

Figura 30 – Imagem original do *dataset* próprio.



Fonte: O Autor (2022)

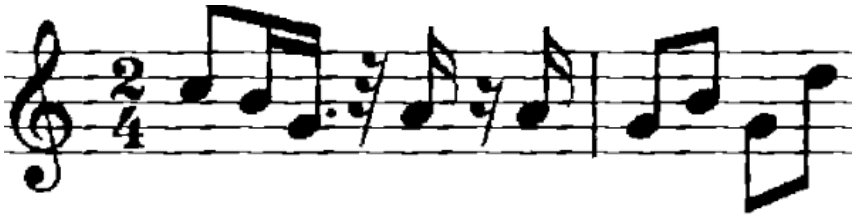
Inicialmente o sistema foi executado apenas para as configurações do filtro de binarização e do grau de rotação da imagem, gerando os resultados da Tabela 4. Notou-se que para o *dataset* PRIMUS não houve diferença de resultado, pois as imagens já eram de boa qualidade e alinhadas. No caso das imagens geradas pelo *software* Noteflight, a melhor execução foi com a binarização de Otsu e um grau de rotação de 0,25, atingindo 75,79% de acerto (191/252 símbolos).

Tabela 4 – Resultados de classificação com binarização x grau de rotação.

Pré-processamento		**Datasets**	
Binarização	**Grau de rotação**	**PRIMUS**	**Próprio**
Global	0,25	97,75	60,32
Global	0,15	97,75	67,46
Global	0,05	97,75	73,81
Otsu	0,25	97,75	75,79
Otsu	0,15	97,75	73,81
Otsu	0,05	97,75	75,00

Fonte: O Autor (2022)

Figura 31 – Imagem do *dataset* próprio resultante da binarização de Otsu x grau de rotação de 0,25.



Fonte: O Autor (2022)

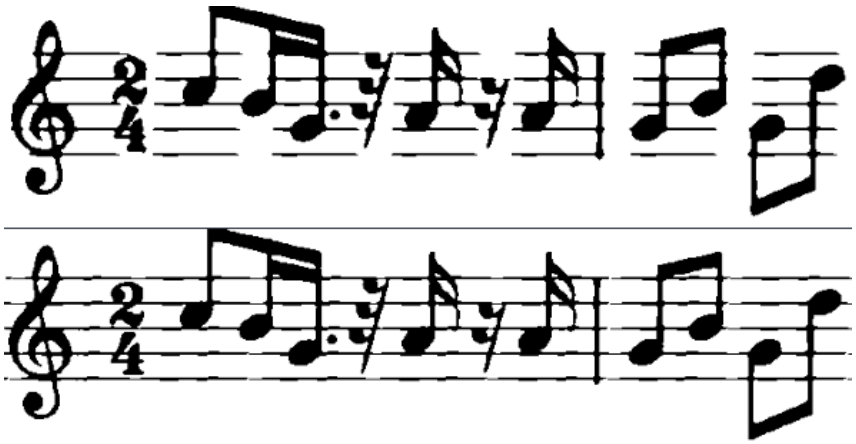
Após a definição do método de binarização com melhor resultado, foi feita uma nova execução considerando também o filtro Gaussiano com diferentes tamanhos de máscara, conforme resultado da Tabela 5. O melhor percentual de classificação obteve 77,38% de acerto (195/252 símbolos), utilizando uma máscara de 3x3 para o filtro Gaussiano. Máscaras maiores causaram a remoção das linhas mais finas das pautas e também a junção de elementos muito próximos, resultando em erros no processo de classificação. Também foi possível perceber que com o filtro Gaussiano os objetos da imagem ficam com bordas mais nítidas, conforme Figura 32.

Tabela 5 – Resultados de classificação com binarização x filtro Gaussiano.

Pré-processamento			Datasets	
Binarização	**Grau de rotação**	**Máscara filtro Gaussiano**	**PRIMUS**	**Próprio**
Otsu	0,25	7x7	94,21	64,29
Otsu	0,25	5x5	97,75	74,60
Otsu	0,25	3x3	97,43	77,38

Fonte: O Autor (2022)

Figura 32 – Imagem do *dataset* próprio. Resultado do filtro Gaussiano 7x7 (cima) e resultado do filtro Gaussiano 3x3 (baixo).



Fonte: O Autor (2022)

Também foi feito o experimento com operações morfológicas de erosão e dilatação em conjunto com a binarização de Otsu. Nessas condições, foi obtido um melhor resultado com um *kernel* de tamanho 3x3, atingindo 72,62% de acerto (183/252 símbolos). Em comparação com um *kernel* de 7x7 obteve-se uma deformação muito grande dos objetos da imagem, os quais tiveram sua área aumentada, preenchendo espaços em branco erroneamente.

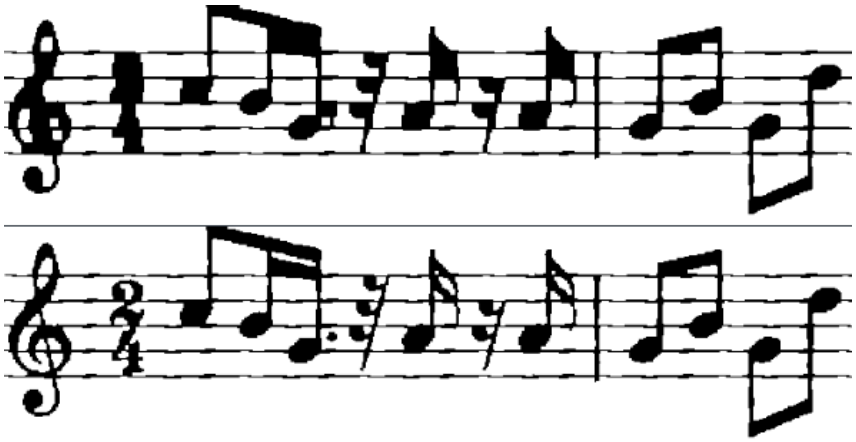
Tabela 6 – Resultados de classificação com binarização x operações morfológicas de erosão e dilatação.

Pré-processamento			Datasets	
Binarização	**Grau de rotação**	**Kernel erosão e dilatação**	**PRIMUS**	**Próprio**
Otsu	0,25	7x7	77,81	44,05
Otsu	0,25	5x5	96,46	63,89
Otsu	0,25	3x3	98,07	72,62

Fonte: O Autor (2022)

Por fim, foi feito o experimento utilizando a binarização, filtro Gaussiano e operações de erosão e dilatação. O resultado obtido foi muito parecido com a Figura 33, portanto, notou-se que para os *datasets* selecionados, apenas a binarização de Otsu já possibilita uma boa filtragem e o filtro Gaussiano com máscara de 3x3 auxilia na melhora da qualidade de borda dos objetos da imagem.

Figura 33 – Imagem do *dataset* próprio. Resultado da erosão/dilatação 7x7 (cima) e resultado da erosão/dilatação 3x3 (baixo).



Fonte: O Autor (2022)

Tabela 7 – Resultados de classificação com binarização x filtro Gaussiano 3x3 x operações morfológicas de erosão e dilatação.

Pré-processamento				**Datasets**	
Binarização	**Grau de rotação**	**Máscara filtro Gaussiano**	**Kernel erosão e dilatação**	**PRIMUS**	**Próprio**
Otsu	0,25	3x3	7x7	78,78	43,25
Otsu	0,25	3x3	5x5	96,46	61,90
Otsu	0,25	3x3	3x3	98,07	72,62

Fonte: O Autor (2022)

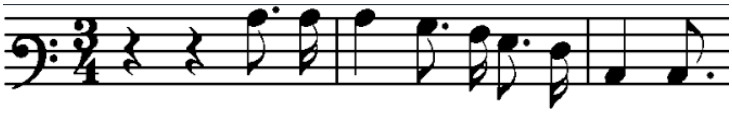
6.2 ERROS DE CLASSIFICAÇÃO

Para entender melhor os tipos de erros de classificação, neste capítulo foi feita uma análise mais detalhada dos arquivos `.misclassified` gerados. A análise foi feita de forma manual e verificando cada arquivo gerado de cada execução que obteve os melhores resultados de classificação.

Para o *dataset* PRIMUS, o erro mais comum foi na classificação das notas das pautas de Fá, onde o classificador apontou problemas no entendimento da semântica da pauta (pautas de Fá possuem notas com tonalidade diferente das pautas de Sol), pois apesar do erro das notas houve o acerto nos tempos (bandeirolas).

Apesar do modelo utilizado neste trabalho ter sido treinado através do *dataset* PRIMUS, a quantidade de pautas em Fá existentes neste *dataset* eram bem menores que a quantidade de pautas em Sol, e isso pode ter gerado um modelo mais especializado em classificar pautas com a clave de Sol. Os exemplos da Figura 34 e Figura 35 demonstram o erro de classificação mais comum analisado. Vale ressaltar que este tipo de erro foi o único ocorrido, explicando o bom resultado de classificação obtido com o *dataset* PRIMUS. Erros para a classificação dos compassos e dos tempos das notas e pausas não aconteceram.

Figura 34 – Pauta do *dataset* PRIMUS resultante da binarização de Otsu e filtro Gaussiano 3x3.



Fonte: O Autor (2022)

Figura 35 – Arquivo dos erros de classificação da pauta de Fá do *dataset* PRIMUS.

images > primus-dataset > 000101654-1_1_1.misclassified

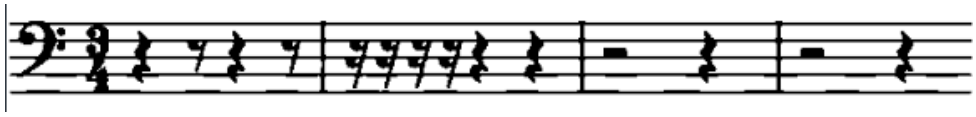
You, 3 weeks ago | 1 author (You)

```
1  {"position": 6, "expected_word": "note-C4_sixteenth", "classified_word": "note-E4_sixteenth"}
2  {"position": 8, "expected_word": "note-C4_quarter", "classified_word": "note-E4_quarter"}
3  {"position": 9, "expected_word": "note-B3_eighth.", "classified_word": "note-D4_eighth."}
4  {"position": 15, "expected_word": "note-C3_eighth.", "classified_word": "note-C3_eighth"}
5
```

Fonte: O Autor (2022)

Já em relação ao *dataset* próprio, houveram erros na classificação dos tempos das pausas, na classificação de alguns compassos como 6/8, 6/4 e 4/4, na identificação das notas (mas com acerto dos tempos) e um erro geral na classificação das pausas, onde a pauta em questão era composta apenas por descansos, conforme exemplo da Figura 36 e Figura 37. Esse tipo de composição não é comum, pois para se ter um compasso inteiro com ausência de notas, basta usar um único símbolo de pausa que obedeça o tempo total daquele compasso, portanto, é provável que pautas deste tipo (apenas com pausas) não estavam presentem no *dataset* PRIMUS.

Figura 36 – Pauta do *dataset* próprio resultante da binarização de Otsu e filtro Gaussiano 3x3.



Fonte: O Autor (2022)

Figura 37 – Arquivo dos erros de classificação da pauta de Fá com pausas do *dataset* próprio.

images > user-generated > staff_13.misclassified

You, 4 weeks ago | 1 author (You)

```
1  {"position": 2, "expected_word": "timeSignature-3/4", "classified_word": "timeSignature-2/4"}
2  {"position": 6, "expected_word": "rest-eighth", "classified_word": "note-E3_eighth"}
3  {"position": 8, "expected_word": "rest-sixteenth", "classified_word": "note-E3_sixteenth"}
4  {"position": 9, "expected_word": "rest-sixteenth", "classified_word": "note-E3_sixteenth"}
5  {"position": 10, "expected_word": "rest-sixteenth", "classified_word": "note-E3_sixteenth"}
6  {"position": 11, "expected_word": "rest-sixteenth", "classified_word": "note-E3_sixteenth"}
7
```

Fonte: O Autor (2022)

Considerando a soma total dos símbolos das pautas próprias, o sistema obteve um percentual de acerto em torno dos 75% a 77%. Os resultados foram muito inferiores comparados aos obtidos com o *dataset* PRIMUS, que chegaram a quase 98% de acerto na classificação dos símbolos. Estes resultados estão muito associados ao modelo utilizado para a predição dos símbolos musicais, que foi treinado utilizando o *dataset* PRIMUS, o qual possui pautas de boa

qualidade e totalmente alinhadas. O *dataset* próprio foi gerado com qualidade reduzida propositalmente, com a finalidade de avaliação do modelo de classificação em condições diferentes das que foi treinado. Em resumo, notou-se que a filtragem pode ajudar a atingir valores mais altos de classificação, contudo, existe a limitação imposta pelo modelo treinado utilizado.

De forma geral e considerando os parâmetros com os melhores resultados de percentual de acerto (filtro Gaussiano com máscara 3x3 em conjunto com binarização de Otsu), para o *dataset* PRIMUS o sistema classificou corretamente 303 de 311 símbolos, resultando em um acerto de classificação de 97,43% e gerando 9 arquivos MIDI, isto é, 9 imagens de pautas tiveram 100% dos seus símbolos classificados corretamente. Para o *dataset* próprio o sistema classificou corretamente 195 de 252 símbolos, resultando em um acerto de classificação de 77,28% e gerando 3 arquivos MIDI, isto é, 3 imagens de pautas tiveram 100% dos seus símbolos classificados corretamente. Com isso concluiu-se que 5 pautas do *dataset* PRIMUS e 11 pautas do *dataset* próprio tiveram algum tipo de erro de classificação.

7 CONSIDERAÇÕES FINAIS

Este trabalho, inicialmente, resumiu alguns aspectos da estrutura musical de partituras CWMN monofônicas, esclarecendo pontos importantes desta linguagem musical. Através de imagens, os principais elementos que compõem uma partitura foram exemplificados, contextualizando diferentes formas e métodos de utilização do material musical. Dentre as diferentes formas de imagens de partituras, concluiu-se que, atualmente, partituras digitais são mais utilizadas em estudos que envolvem a área de reconhecimento de imagens, pois possuem melhor qualidade e maior acervo disponível. Pautas manuscritas trazem uma complexidade muito maior para um classificador de símbolos, pois pode existir diferentes formatos e tamanhos de símbolos, considerando a caligrafia de cada compositor.

Estudos de conceitos, arquiteturas de sistemas OMR e *datasets* de partituras musicais foram descritos e comparados. Algumas técnicas mais comuns foram abordadas, objetivando a estruturação do protótipo deste trabalho, que é fundamentado em uma arquitetura que utiliza binarização, filtragem de ruídos e alinhamento de pauta para o pré-processamento da imagem, redes neurais para a classificação dos símbolos musicais e o formato MIDI para a representação final do reconhecimento. O uso do modelo treinado pela rede neural discutida simplificou e acelerou o desenvolvimento do protótipo, agregando os resultados positivos já obtidos por outros autores.

A linguagem Python e as ferramentas OpenCV e TensorFlow foram escolhidas para compor o projeto, principalmente pela simplicidade e pelo material de suporte disponível para consulta. Estas ferramentas possuem uma grande comunidade e também são as mais citadas nas áreas que abrangem o protótipo desenvolvido.

A composição das pautas próprias e de seus arquivos semânticos foi feita de forma manual e criativa. O processo foi bastante demorado e a utilização do sistema web Noteflight foi essencial, facilitando a criação, pois a maioria dos *softwares* de composição possuem limitações de funcionalidade ou são pagos. O estudo e entendimento do vocabulário semântico também foi de extrema importância para a criação das pautas próprias, fazendo-se necessária a análise das pautas do *dataset* PRIMUS para que a criação dos arquivos semânticos próprios não fosse feita de forma equivocada.

O desenvolvimento deste trabalho teve como objetivo, além da contribuição tecnológica para a área da música, o estudo e aquisição de conhecimento sobre as tecnologias mais atuais e citadas nas áreas de processamento e reconhecimento de imagens de partituras digitais. Visando oportunidades de trabalhos futuros, podem ser estudadas adaptações e melhorias da classificação de pautas próprias, focando no treino de um novo modelo através do *dataset* Camera-PRIMUS (PRIMUS-DATASET, 2018), que possui imagens de pautas de baixa qualidade.

O formato final de codificação também pode ser explorado utilizando a codificação em MusicXML. Com formatos diferentes o sistema pode se adaptar a novos objetivos, como o de compartilhamento e edição de partituras reconhecidas através de sistemas de terceiros, como o próprio Noteflight. Espera-se que a pesquisa realizada complemente conhecimentos e contribua para a elaboração de projetos futuros.

REFERÊNCIAS

ABRAHAMS, S. *et al.* **TensorFlow for Machine Intelligence: A Hands-On Introduction to Learning Algorithms**. [S.l.]: Bleeding Edge Press, 2016. 305 p. ISBN 978-1-93-990235-1.

BFM. OpenCV, 2022. Brute-Force Matcher. Disponível em: <https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html>. Acesso em: 09 ago. 2022.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV**. [S.l.]: O'Reilly, 2008. 577 p. ISBN 978-0-59-651613-0.

BURTSEV, S.; KUZMIN, Y. An efficient flood-filling algorithm. **Computers and Graphics**, v. 17, n. 5, p. 549–561, 1993. ISSN 0097-8493. Doi:10.1016/0097-8493(93)90006-U.

BYRD, D.; SIMONSEN, J. Towards a standard testbed for optical music recognition: Definitions, metrics, and page images. **Journal of New Music Research**, v. 44, Jul. 2015. Doi:10.1080/09298215.2015.1045424.

CALVO-ZARAGOZA, J. *et al.* Avoiding staff removal stage in optical music recognition: application to scores written in white mensural notation. **Formal Pattern Analysis & Applications**, v. 18, Sep. 2014. Doi:10.1007/s10044-014-0415-5.

CALVO-ZARAGOZA, J.; JR., J. H.; PACHA, A. Understanding optical music recognition. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, p. 35, Jul. 2020. ISSN 0360-0300. Doi:10.1145/3397499.

CALVO-ZARAGOZA, J.; ONCINA, J. Recognition of pen-based music notation: The homus dataset. p. 3038–3043, 2014. Doi:10.1109/ICPR.2014.524.

CALVO-ZARAGOZA, J.; RIZO, D. End-to-end neural optical music recognition of monophonic scores. **Applied Sciences**, v. 8, n. 4, Apr. 2018. ISSN 2076-3417. Doi:10.3390/app8040606.

CARDOSO, J.; REBELO, A. Robust staffline thickness and distance estimation in binary and gray-level music scores. p. 1856–1859, Ago. 2010. Doi:10.1109/ICPR.2010.458.

CHAKI, J.; DEY, N. A beginner's guide to image pre-processing techniques. Jun. 2018. Doi:10.1201/9780429441134.

CSURKA, G. *et al.* Visual categorization with bags of keypoints. **Work Stat Learn Comput Vision, ECCV**, Jan. 2004. Disponível em: <https://www.researchgate.net/publication/228602850_Visual_categorization_with_bags_of_keypoints>. Acesso em: 08 Dez. 2022.

DALITZ, C. *et al.* A comparative study of staff removal algorithms. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 30, n. 5, p. 753–766, 2008. Doi:10.1109/TPAMI.2007.70749.

FILHO, O. M.; NETO, H. V. **Processamento Digital de Imagens**. 1. ed. [S.l.]: Brasport, 1999. 331 p. ISBN 8574520098.

FLOODFILL. OpenCV, 2022. Miscellaneous Image Transformations. Disponível em: <https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html>. Acesso em: 09 ago. 2022.

FUJINAGA, I. Optical music recognition using projections. p. 65, Sep. 1988. Disponível em: <https://escholarship.mcgill.ca/downloads/gb19f701g>. Acesso em: 01 abr. 2022.

____. Staff detection and removal. **Visual Perception of Music Notation: On-Line and Off-Line Recognition**, p. 1–39, Jan. 2004. Doi:10.4018/978-1-59140-298-5.ch001.

FURUKAWA, E. S.; PEDRINI, H. Leitura de partituras em imagens digitais. Instituto de computação da universidade estadual de campinas, Jan. 2021. Disponível em: <https://www.ic.unicamp.br/~reltech/PFG/2020/PFG-20-26.pdf>. Acesso em: 01 abr. 2022.

GOECKE, R. Building a system for writer identification on handwritten music scores. p. 7, Jun. 2003. Disponível em: <https://www.researchgate.net/publication/228699586_Building_a_system_for_writer_identification_on_handwritten_music_scores>. Acesso em: 01 abr. 2022.

GOMEZ, A. A.; SUJATHA, C. N. Optical music recognition: Staffline detection and removal. **International Journal of Application or Innovation in Engineering & Management (IJAIEM)**, v. 6, n. 5, p. 048–058, Mai. 2017. ISSN 2319-4847. Disponível em: <https://www.ijaiem.org/Volume6Issue5/IJAIEM-2017-05-18-19.pdf>. Acesso em: 01 abr. 2022.

GRANA, C. *et al.* **A Fast Approach for Integrating ORB Descriptors in the Bag of Words Model**. [S.l.: s.n.], 2013. v. 8667. Doi:10.1117/12.2008460.

HLT. OpenCV, 2022. Hough Line Transform. Disponível em: <https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html>. Acesso em: 09 ago. 2022.

ILLINGWORTH, J.; KITTLER, J. A survey of the hough transform. **Computer Vision, Graphics, and Image Processing**, v. 44, n. 1, p. 87–116, 1988. ISSN 0734-189X. Doi:10.1016/S0734-189X(88)80033-1.

JAKUBOVIć, A.; VELAGIć, J. Image feature matching and object detection using brute-force matchers. **2018 International Symposium ELMAR**, p. 83–86, Nov. 2018. Doi:10.23919/ELMAR.2018.8534641.

JAVACV. GitHub, 2022. Java interface to OpenCV, FFmpeg, and more. Disponível em: <https://github.com/bytedeco/javacv>. Acesso em: 09 ago. 2022.

JESUS, E. O.; JR., R. C. A utilização de filtros gaussianos na análise de imagens digitais. **Proceeding Series of the Brazilian Society of Applied and Computational Mathematics**, v. 3, p. 7, 2015. Doi:10.5540/03.2015.003.01.0118.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. Mar. 1995. Disponível em: <https://www.ijcai.org/Proceedings/95-2/Papers/016.pdf>. Acesso em: 09 ago. 2022.

MARGARIDA, T.; SILVA, A. G. Reconhecimento automático de símbolos em partituras musicais. Universidade do Estado de Santa Catarina (UDESC), p. 8, 2013. Disponível em: <https://www.yumpu.com/pt/document/read/12695349/reconhecimento-automatico-de-simbolos-em-partituras-puc-rio>. Acesso em: 01 abr. 2022.

MAURICENZ, J. Interpres, um protótipo para reconhecimento de partitura: identificando a forma dos símbolos musicais. Universidade Regional de Blumenau, Jul. 2013. Disponível em: <http://campeche.inf.furb.br/tccs/BCC/2013-I/TCC2013-1-19-VF-JonathanMauricenzy.pdf>. Acesso em: 01 abr. 2022.

MEI. GitHub, 2022. Music Encoding Initiative. Disponível em: <https://github.com/bytedeco/javacv>. Acesso em: 09 ago. 2022.

MIDI. MIDI Association, 2022. Musical Instrument Digital Interface. Disponível em: <https://midi.org/specifications>. Acesso em: 09 ago. 2022.

MIDIUTIL. PyPI, 2022. A pure python library for creating multi-track MIDI files. Disponível em: <https://pypi.org/project/MIDIUtil/>. Acesso em: 09 ago. 2022.

MUSESCORE. Musescore, 2022. The world's most popular notation software. Disponível em: <https://musescore.org/en>. Acesso em: 09 ago. 2022.

MUSICXML. MakeMusic, 2022. The standard open format for exchanging digital sheet music. Disponível em: <https://www.musicxml.com/>. Acesso em: 09 ago. 2022.

NA, I.; KIM, S.; NQUYEN, T. A robust staff line height and staff line space estimation for the preprocessing of music score recognition. **Journal of Internet Computing and Services**, v. 16, p. 29–37, Fev. 2015. Doi:10.7472/jksii.2015.16.1.29.

NG, K. *et al.* Embracing the composer: Optical recognition of handwritten manuscripts. Out. 1999. Disponível em: <https://www.researchgate.net/publication/315840868_Embracing_the_Composer_Optical_Recognition_of_Handwritten_Manuscripts>. Acesso em: 01 abr. 2022.

NOBLE, W. S. What is a support vector machine? **Nature Biotechnology**, v. 24, p. 62–66, 2006. Doi:10.1038/nbt1206-1565.

NOTEFLIGHT. Noteflight, 2022. A Hal Leonard Company. Disponível em: <https://www.noteflight.com/>. Acesso em: 09 set. 2022.

NOVOTNý, J.; POKORNý, J. Introduction to optical music recognition: Overview and practical challenges. **CEUR Workshop Proceedings**, v. 1343, p. 65–76, Jan. 2015. Disponível em: <https://www.researchgate.net/publication/281786658_Introduction_to_optical_music_recognition_Overview_and_practical_challenges>. Acesso em: 01 abr. 2022.

OMR-DATASETS. GitHub, 2022. Collection of datasets used for Optical Music Recognition. Disponível em: <https://apacha.github.io/OMR-Datasets>. Acesso em: 01 abr. 2022.

OMR-SISTEMA. GitHub, 2022. Sistema OMR para reconhecimento de partituras musicais monofônicas. Disponível em: <https://github.com/leandro-colle/omr-sistema>. Acesso em: 22 out. 2022.

OPENCV. OpenCV, 2022. OpenCV-Python Tutorials. Disponível em: <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>. Acesso em: 01 abr. 2022.

OPENCV-ORG. OpenCV, 2022. Open Source Computer Vision Library. Disponível em: <https://opencv.org>. Acesso em: 11 ago. 2022.

ORB. OpenCV, 2022. Oriented FAST and Rotated BRIEF. Disponível em: <https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html>. Acesso em: 09 ago. 2022.

OTSU, N. A threshold selection method from gray-level histograms. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 9, n. 1, p. 62–66, 1979. Doi:10.1109/TSMC.1979.4310076.

PACHA, A.; CALVO-ZARAGOZA, J. Optical music recognition in mensural notation with region-based convolutional neural networks. Sep. 2018. Doi:10.13140/RG.2.2.28624.40965.

PACHA, A.; CALVO-ZARAGOZA, J.; HAJIC, J. Learning notation graph construction for full-pipeline optical music recognition. **International Society for Music Information Retrieval (ISMIR)**, 2019. Doi:10.5281/zenodo.3527744.

PINTO, T. *et al.* Music score binarization based on domain knowledge. v. 6669, p. 700–708, Jun. 2011. Doi:10.1007/978-3-642-21257-4_87.

PRIMUS-DATASET. Pattern Recognition and Artificial Intelligence Group, 2018. The Printed Images of Music Staves (PrIMuS) dataset. Disponível em: <https://grfia.dlsi.ua.es/primus>. Acesso em: 01 abr. 2022.

REBELO, A.; CAPELA, G.; CARDOSO, J. Optical recognition of music symbols - a comparative study. **International Journal on Document Analysis and Recognition (IJDAR)**, v. 13, p. 19–31, Mar. 2010. Doi:10.1007/s10032-009-0100-1.

REBELO, A. *et al.* Optical music recognition: State-of-the-art and open issues. **International Journal of Multimedia Information Retrieval**, v. 1, Out. 2012. Doi:10.1007/s13735-012-0004-6.

SCIKIT-IMAGE. Scikit-Image, 2022. Image processing in Python. Disponível em: <https://scikit-image.org/>. Acesso em: 08 ago. 2022.

SHATRI, E.; FAZEKAS, G. Optical music recognition: State of the art and major challenges. **ArXiv**, abs/2006.07885, 2020. Doi:10.48550/arXiv.2006.07885.

SILVA, G. Ântonny de Sousa da. Reconhecimento automático de partituras manuscritas utilizando svm e orb. Universidade Federal do Maranhão, Nov. 2017. Disponível em: <https://monografias.ufma.br/jspui/handle/123456789/3563>. Acesso em: 01 abr. 2022.

SVM. OpenCV, 2022. Support Vector Machines. Disponível em: <https://docs.opencv.org/4.x/d1/d73/tutorial_introduction_to_svm.html>. Acesso em: 09 ago. 2022.

SYNTHESIA. Synthesia LLC, 2022. A fun way to learn how to play the piano. Disponível em: <https://synthesiagame.com/>. Acesso em: 09 ago. 2022.

TENSORFLOW. TensorFlow, 2022. A complete open source platform for machine learning. Disponível em: <https://www.tensorflow.org/api_docs/python/tf/>. Acesso em: 01 abr. 2022.

TENSORFLOW-ORG. Google, 2022. An end-to-end open source machine learning platform. Disponível em: <https://www.tensorflow.org/>. Acesso em: 11 ago. 2022.

TF-END-TO-END. GitHub, 2022. TensorFlow code to perform end-to-end Optical Music Recognition on monophonic scores through Convolutional Recurrent Neural Networks and CTC-based training. Disponível em: <https://github.com/OMR-Research/tf-end-to-end>. Acesso em: 01 abr. 2022.

XML. W3C, 2022. Extensible Markup Language. Disponível em: <https://www.w3.org/XML/>. Acesso em: 09 ago. 2022.