

# SCOREREADER – PROTÓTIPO PARA RECONSTRUÇÃO DE PARTITURAS, ATRAVÉS DE ELEMENTOS IDENTIFICADOS A PARTIR DE IMAGENS

Alan Soares Carneiro, Marcel Hugo – Orientador

Curso de Bacharel em Ciência da Computação  
Departamento de Sistemas e Computação  
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

ascarneiro@furb.br, marcel@furb.br

**Resumo:** Este artigo apresenta a extensão do protótipo *Interpres* para reconstrução de partitura a partir de símbolos identificados em uma imagem. Para desenvolvimento do protótipo *ScoreReader* foi utilizado o conceito de aplicações cliente-servidor. Na parte cliente da aplicação foi utilizada a linguagem de programação Java, enquanto que no Servidor foi utilizado Python. Para segmentação dos símbolos e identificação das formas presentes na imagem foi utilizada a biblioteca *OpenCV* e para classificação dos símbolos, foi utilizado o algoritmo *K-Nearest Neighbors* disponibilizado pela biblioteca *scikit-learn* em Python. Para treinamento do classificador de símbolos musicais, o protótipo disponibiliza interface específica para rotulação e treinamento. Após segmentados e classificados, os símbolos são convertidos para linguagem de notação musical ABC e compiladas através de compiladores open source específicos para formatos PDF, MIDI e MUSICXML. O protótipo comportou-se bem para imagens de partituras digitais com instrumentos de uma única voz. Foram destacadas algumas limitações nas etapas segmentação dos símbolos e detecção da altura das notas, ambas dependentes da nitidez da imagem. Na etapa de segmentação a falha ocorria porque o símbolo é dividido em mais partes dificultando a classificação. Na etapa de detecção da altura porque não eliminava as hastes do símbolo corretamente impossibilitando detecção no nome da nota.

**Palavras-chave:** OMR. Partituras. Inteligência artificial. K-Nearest Neighbors. *Interpres*. Processamento de imagens.

## 1 INTRODUÇÃO

Imaginando um cenário quase que comum nas orquestras, um músico ou maestro recebe em mãos uma partitura que precisa ser transcrita para outro instrumento que não o original da partitura recebida. Esta necessidade gera retrabalho ao músico, que apesar de poder utilizar softwares e ferramentas de edição disponíveis, precisa redigitar toda a partitura e somente ao fim utilizar uma funcionalidade que altera a altura das notas musicais para leitura de outro instrumento (MARGARIDA, 2009). Seufert (2009, p. 3), também nos conduz a um outro cenário pouco mais distante, mas relacionado, voltado à preservação de partituras musicais. Ele expõe como ao longo dos anos tem se perdido muito da cultura musical devido à má preservação de certos documentos impressos de partituras, e aponta como a digitação manual de partituras por um copista é dispendiosa em tempo e sucessível a erros.

Para resolução destas duas problemáticas, aplicam-se os sistemas OMR (Optical Music Recognition), que tem como função principal a transformação de folhas de músicas impressas em um formato legível por computador (CASTRO, 2014). Segundo Seufert (2010, p. 2), aliado aos sistemas OMR é necessário empregar novas técnicas para o reconhecimento dos elementos musicais; técnicas como aprendizagem automática e aprendizagem de máquina que auxiliarão no reconhecimento e classificação dos símbolos de forma mais precisa. Tais técnicas contribuirão com a preservação da cultura musical de modo inovador.

Com foco no ensino musical a deficientes visuais, o protótipo *Interpres* (MAURICENZ, 2013) teve como objetivo o desenvolvimento de um software OMR em linguagem de programação Java, capaz de analisar uma imagem de partitura e fornecer o reconhecimento dos símbolos contidos. O protótipo limitou-se a partituras simples, atuando nas etapas de remoção de linhas, segmentação e reconhecimento dos símbolos, empregando técnica de descritores de forma para classificação.

O presente trabalho desenvolveu a extensão do protótipo de Mauricenz (2013), cobrindo algumas falhas apontadas pelo autor e contribuindo ao adicionar novas técnicas para reconhecimento. Os objetivos específicos propostos foram: remoção das linhas em uma imagem em qualquer posição através de biblioteca externa específica; emprego de aprendizagem de máquina para melhoria do processo de classificação; transcrição dos símbolos reconhecidos para a linguagem intermediária Notação ABC; e compilação de código ABC para formatos MUSICXML, PDF e MIDI. Os resultados apresentados são compostos da análise de imagens obtidas da internet, e algumas partituras elaboradas pelo autor através de um editor de partituras.

## 2 FUNDAMENTAÇÃO TEÓRICA

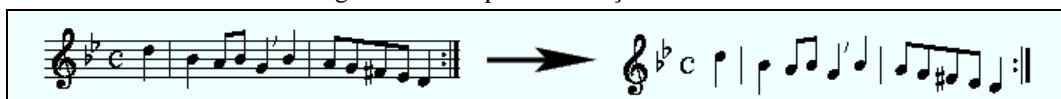
Este capítulo é composto pelos principais conceitos e técnicas estudados e aplicados no desenvolvimento deste trabalho. A seção 2.1 descreve um pouco sobre processamento de imagens e sua utilização. Nele é abordado o OPENCV (Open Source Computer Vision Library), principal biblioteca de processamento de imagens utilizada no trabalho assim como a biblioteca Musicstaves (DALITZ et al. 2008), principal responsável pela remoção das linhas horizontais da pauta musical. Na seção 2.2 é oferecida uma breve introdução ao aprendizado por máquina e os três principais tipos de aprendizagem, com destaque ao aprendizado supervisionado. Nela é apresentada também uma breve explicação sobre o algoritmo KNN (K-NEAREST NEIGHBOR - algoritmo do vizinho mais próximo), por ser o algoritmo utilizado para o reconhecimento de símbolos neste trabalho. É introduzido o conjunto de dados MUSCIMA++ (HAJIC JUNIOR; PECINA, 2017), que corresponde a um conjunto de entradas de figuras musicais manuscritas já rotuladas e que podem ser utilizadas para o treinamento do algoritmo KNN, sendo possível ainda rotular tais dados de treinamento no próprio protótipo. Por fim na seção 2.3 é feita uma breve introdução à notação ABC, que é a linguagem intermediária utilizada para reconstrução da notação musical. Nela são abordados também os principais compiladores de notação ABC utilizados que possibilitam a compilação da notação reescrita nos formatos PDF, MIDI e MUSICXML.

### 2.1 PROCESSAMENTO DE IMAGENS COM OPENCV E MUSICSTAVES

Segundo Maillard (2001, p. 1), o processamento de imagens envolve o desenvolvimento de técnicas e algoritmos a fim de melhorar ou modificar o aspecto visual das imagens. Tudo se inicia com a captura de uma imagem, a qual normalmente corresponde à iluminação que é refletida na superfície dos objetos. Uma vez capturada, a imagem passa por um processo de digitalização onde é representada de uma forma apropriada para o tratamento computacional. O primeiro passo efetivo do processamento de imagens é conhecido como pré-processamento, o qual envolve passos como remoção de ruídos e correções (Queiroz; Gomes, 2006). Segundo a Intel (2019). O OpenCV é uma biblioteca de processamento de imagens e aprendizado de máquina de código fonte aberto. Concebida para fornecer infraestrutura comum para aplicativos de visão computacional, possui mais de 2500 algoritmos clássicos e avançados. Tais algoritmos podem ser utilizados por exemplo para detectar rostos, identificar objetos, rastrear movimentos dentre muitas outras funcionalidades. Possui interfaces para diversas linguagens de programação desde C++, Python, Java dentre outras.

Segundo Dalitz et al. (2008, p. 1), o Musicstaves é um conjunto de ferramentas escrito em linguagem de programação Python e oferecido na forma de plugin do Gamera Project (DALITZ; DROETTBOOM; FUJINAGA, 2019). Tem como objetivo experimentar diferentes métodos de remoção de linhas da imagem de uma partitura musical e fornece uma grande variedade de métodos, uma vez que nem sempre um mesmo método poderá ser adequado para todas as circunstâncias ou imagens. Pode ser utilizado para partituras impressas, manuscritas, tablaturas e para notação em música antiga. O método de remoção de linhas ainda permite que ao remover as linhas, a posição que representa a altura original de cada uma das linhas seja recuperada com intuito de utilizar para detecção de altura dos sons das figuras musicais contidas nestas linhas (DALITZ et al., 2008). Na Figura 1 é possível ver a ação do algoritmo.

Figura 1– Exemplo de remoção das linhas



Fonte: Dalitz et al. (2008, p. 1).

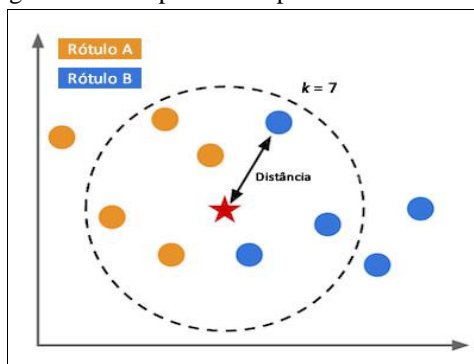
### 2.2 APRENDIZADO DE MÁQUINA, KNN (K-NEAREST NEIGHBOR) E CONJUNTO DE DADOS MUSCIMA++

Segundo Stange (2011, p. 1), o termo aprendizado de máquina ou *Machine Learning*, refere-se ao funcionamento de sistemas computacionais capazes de aprender ou ter seu comportamento modificado através de experiências acumuladas durante sua operação. Para Russell e Norvig (2013, p. 606), pode-se destacar três tipos principais de aprendizado: aprendizado supervisionado, aprendizado por reforço e aprendizado não supervisionado. Na aprendizagem não supervisionada o agente recebe um conjunto de padrões de entrada, porém não lhe é fornecido nenhum feedback ou pista do que se trata cada entrada. Cabe ao algoritmo agrupar estas entradas tendo como base características comuns. Na aprendizagem por reforço o agente aprende a partir de uma série de reforços, que seriam em outras palavras recompensas ou punições para determinada ação. Cabe ao agente, de acordo com o objetivo a ser alcançado, decidir em receber uma recompensa ou punição que o levará ao cumprimento do objetivo. Na aprendizagem supervisionada o agente recebe tanto padrões de entrada como suas respectivas saídas e a partir destas aprende a função que faz o mapeamento desta entrada para as saídas. A intenção é que, dependendo da quantidade de exemplos recebidos, o agente seja capaz de prever outras saídas corretas mesmo sem tê-las conhecido. No tipo de aprendizagem supervisionada está fundamentado o algoritmo de reconhecimento de padrões utilizado para este trabalho.

Segundo Mariz (2017, p. 14), o algoritmo KNN é um dos algoritmos mais famosos para o reconhecimento de padrões, por ser um dos mais simples e efetivos métodos de classificação e regressão. É um método de predição que

utiliza a distância atual da amostra ou no caso o elemento a ser reconhecido, em relação aos seus vizinhos mais próximos como o nome sugere. Com base nesta distância, o algoritmo irá prever ou determinar a que classe ou padrão pertence a amostra. Segundo Cunningham e Delany (2007, p. 1), talvez o KNN seja o algoritmo mais direto do arsenal de técnicas de aprendizado de máquina. De acordo com a documentação oficial do Scikit-learn (PEDREGOSA et al. 2011), que fornece implementações do KNN em linguagem de programação Python, apesar da simplicidade do algoritmo KNN, ele obteve sucesso em muitos problemas de classificação e regressão, incluindo análise e classificação de dígitos manuscritos, sendo este um dos principais motivos da escolha deste algoritmo para o trabalho, dada também sua simplicidade e quantidade de exemplos disponíveis na internet. Na Figura 2 pode-se observar um exemplo de como as classes ficam distribuídas e como o cálculo de distância é feito em relação a estas classes. A implementação do algoritmo KNN da biblioteca permite por meio de parametrização a utilização de diferentes modelos de cálculos para da distância entre os pontos. O padrão caso não seja explicitado é utilizar a métrica ou espaço de Minkowsky.

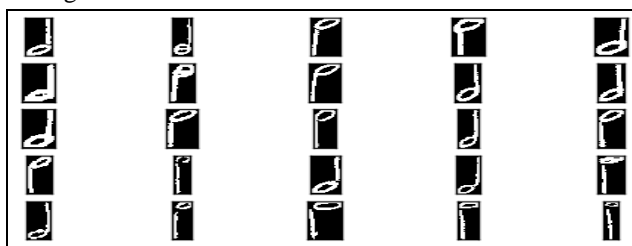
Figura 2– Exemplo de comportamento do KNN



Fonte: Pacheco (2017).

Para treinamento do algoritmo KNN foram utilizados dois tipos de imagens rotuladas, sendo um de tipo impresso e outro manuscrito. As imagens impressas foram extraídas e rotuladas dentro do protótipo. Para o treinamento com imagens manuscritas foi utilizado o conjunto de dados MUSCIMA++. Segundo Hajic Junior e Pecina (2017, p. 1), o MUSCIMA++ é um conjunto de símbolos musicais escritos à mão, convertidos para primitivas internas. Ele contém rotulações de cerca de 90.000 objetos de notação musical, tratando-se de uma extensão ou derivação do CVC-MUSCIMA. Quando as imagens forem extraídas é necessário reconstruí-las ou redesenhá-las pois estão armazenadas na forma de primitivas. Na Figura 3 vê-se alguns exemplos de primitivas que o conjunto de dados MUSCIMA++ possui.

Figura 3– Primitivas reconstruídas do MUSCIMA++



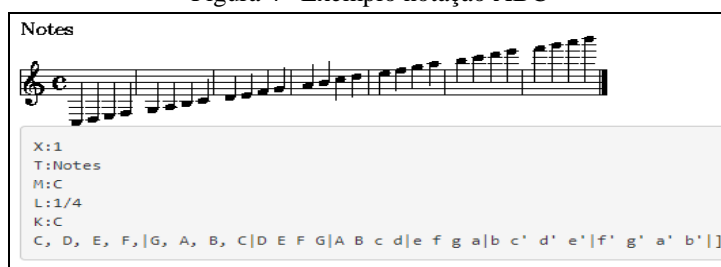
Fonte: Hajic Junior e Pecina (2017).

### 2.3 LINGUAGEM INTERMEDIARIA: NOTAÇÃO MUSICAL ABC

Segundo Gonzato (2003, p. 4), músicos que utilizam computadores podem obter diversos programas para escrever música. A maior parte destes programas utiliza abordagem gráfica, porém neste meio há também os programas com uma abordagem alternativa e que possuem como premissa a utilização de caracteres ASCII como fim de representação de notas e símbolos. Neste caso particular, um interpretador ou compilador traduzirá esta notação para um formato escolhido, com possibilidade de saída PDF, MIDI dentre outros. A Notação ABC utiliza representação textual e tem sido grandemente difundida entre a comunidade de musicistas, dada sua simplicidade e capacidade de representação.

De acordo Lacerda (2009, p. 1), uma música escrita em Notação ABC é estruturada basicamente em Cabeçalho que conterá informações tais como título da partitura, tonalidade da música, compositor dentre outros, enquanto o corpo representa a música propriamente. No corpo serão encontradas ocasionalmente alturas, durações de cada nota ou pausa. Na Figura 4 é mostrado um código escrito utilizando esta notação ABC na parte inferior, e na parte superior da imagem sua respectiva tradução por um interpretador.

Figura 4– Exemplo notação ABC



Fonte: Walshaw (1995).

Para compilação do código escrito em linguagem notação ABC o protótipo utilizou compiladores distribuídos sob licença *Open Source*. Estes compiladores são fornecidos na forma de software livre e foram obtidos através de links no site <http://abcnotation.com/software>. Por meio do protótipo eles são acionados para compilação da notação e geração dos formatos de saídas. A Figura 5 mostra alguns dos compiladores disponíveis para a linguagem e o Quadro 1 fornece uma breve descrição de para que serve cada compilador.

Figura 5 – Compiladores notação ABC

	Nome	Data de modificaç...	Tipo	Tamanho
1	abc2midi.exe	08/09/2019 23:46	Aplicativo	156 KB
2	abc2xml.exe	01/12/2019 16:13	Aplicativo	3.700 KB
3	abcm2ps.exe	08/09/2019 23:46	Aplicativo	455 KB
	midibat	09/09/2019 03:01	Arquivo em Lotes ...	1 KB
	musicxml.bat	09/09/2019 02:27	Arquivo em Lotes ...	1 KB
	svg.bat	09/09/2019 02:27	Arquivo em Lotes ...	1 KB

Fonte: elaborado pelo autor.

Quadro 1 – Descrição dos compiladores para MIDI, SVG e MUSICXML

Identificador	Descrição
1	Compilador de código em notação ABC para MIDI
2	Compilador de código em notação ABC para MUSICXML
3	Compilador de código em notação ABC para SVG

Fonte: elaborado pelo autor.

## 2.4 TRABALHOS CORRELATOS

Nesta seção são apresentados três trabalhos correlatos com aspectos similares ao protótipo desenvolvido. Entre os trabalhos selecionados no Quadro 2 está a dissertação de mestrado de Castro (2014), focado na etapa de detecção e remoção das linhas de pautas musicais, bem como segmentação de símbolos. O trabalho foi aplicado com domínio em imagens de tom cinza. Em sequência no Quadro 3 é apresentado o Sistema autônomo (hardware e software) para reconhecimento de partituras de Jesus (2011). Por último no Quadro 4 é apresentado o trabalho de conclusão de curso de Silva (2017), que tem como objetivo remoção das linhas, detecção dos símbolos e classificação dos símbolos utilizando SVM (Support Vector Machine).

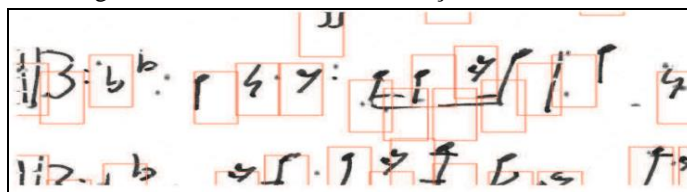
Quadro 2 – Trabalho Correlato 1

Referência	Castro (2014)
Objetivos	Reconhecimento de símbolos musicais em imagem de cinza em partituras manuscritas. Trata-se de um estudo focado nas etapas de remoção das linhas e segmentação dos símbolos.
Principais funcionalidades	Transformar imagens em tom de cinza, detecção e remoção das linhas, segmentação dos símbolos.
Ferramentas de desenvolvimento	Linguagem de programação C, OpenCV.
Resultados e conclusões	Segundo Castro (2014, p. 38), com base nos resultados ele concluiu que seu algoritmo de reconhecimento de remoção das linhas da pauta obteve melhor resultado do que alguns algoritmos de trabalhos correlatos escolhidos por ele, embora seu desempenho ficasse abaixo se comparado utilizando imagens binárias. Quanto à etapa de segmentação dos símbolos, os resultados se apresentaram insatisfatórios dado a precisão média obtida devido ao número de falsas detecções e detecções falhas dependendo do limiar utilizado.

Fonte: elaborado pelo autor.

Na Figura 6 é apresentada uma imagem contendo o resultado após a execução dos algoritmos de remoção das linhas da pauta e segmentação.

Figura 6 – Resultado da identificação dos elementos



Fonte: Castro (2014, p. 37).

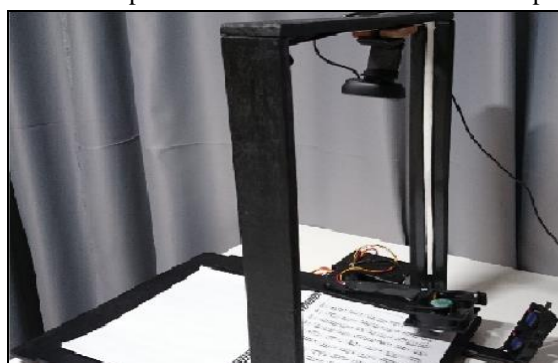
Quadro 3 – Trabalho Correlato 2

Referência	Jesus (2015)
Objetivos	Sistema autônomo (hardware e software) para reconhecimento de partituras, incluindo manipulação de páginas, leitura, interpretação e reprodução em formato MIDI de partituras para instrumentos de uma única mão ou uma só voz.
Principais funcionalidades	Virar Folha, leitura e correção da imagem, binarização da imagem, remoção das linhas da pauta, segmentação dos artefatos ou figuras, reconhecimento das notas, determinar duração, tom e criação de onda senoidal para a pauta e reprodução da música.
Ferramentas de desenvolvimento	Linguagem de programação C# e C (Arduino), MathLab.
Resultados e conclusões	Segundo Jesus (2015, p. 20), a taxa de acerto passou da metade das notas analisadas, gerando em torno de 68% de reconhecimento com pico de 82% e mínimo de 54% para imagens danificadas. Com um ambiente controlado e intervenção e auxílio do usuário o acerto chegou a 90%.

Fonte: elaborado pelo autor.

Na Figura 7 é possível observar como foi construído o protótipo. Nele está posicionado uma câmera na parte superior para fazer a captura da imagem, enquanto na parte inferior é possível visualizar o caderno de música e o dispositivo que realiza a ação de virar a página com auxílio de uma ventoinha.

Figura 7 – Protótipo de sistema autônomo de leitura de partituras



Fonte: Jesus (2015, p. 9).

Quadro 4 – Trabalho Correlato 3

Referência	Silva (2017)
Objetivos	Reconhecimento automático de partituras manuscritas.
Principais funcionalidades	Detecção e remoção das linhas da pauta, segmentação de símbolos musicais, reconhecimento dos símbolos segmentados, Reconstrução da notação musical.
Ferramentas de desenvolvimento	Linguagem e ambiente não especificados, algoritmos ORM (Oriented FAST e Rotated BRIEF) para aquisição de características em uma base de partituras manuscritas e algoritmos de SVM (Support Vector Machine) para classificação dos símbolos.
Resultados e conclusões	Segundo Silva (2017, p. 7), obteve resultados que atingiram 98,05% de acurácia para classificação de 20 classes de símbolos, 100% na etapa de detecção das pautas, 98,28% no processo de remoção das pautas, 86,12% no estágio de identificação de posição das notas e 66,12% no processo de reconhecimento de barras de compasso.

Fonte: elaborado pelo autor.

Na Figura 8 é possível visualizar um fragmento do resultado de uma partitura que passou pelo processo de remoção das linhas da pauta, segmentação e reconhecimento dos símbolos, bem como sua respectiva reconstrução final, no que se percebe alguns erros de identificação de notas.

Figura 8 – Partitura recuperada

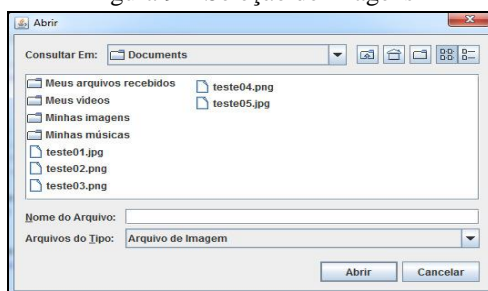


Fonte: Silva (2017, p. 53).

## 2.5 VERSÃO ANTERIOR DO PROTÓTIPO INTERPRES

Segundo Mauricenz (2013, p. 12), o Interpres é um protótipo desenvolvido em linguagem de programação Java capaz de reconhecer símbolos musicais a partir de uma imagem de partitura. O processo de reconhecimento ocorre primeiramente removendo-se as linhas da pauta. Posteriormente isola-se os símbolos musicais através de morfologia matemática e calcula-se descritores de Fourier para caracterizar a forma individual de cada símbolo. Por fim utiliza-se de árvore de decisão para classificação e identificação de cada símbolo. Segundo Mauricenz (2013, p. 54), os resultados demonstraram que o protótipo atingiu cerca de 77% de acerto para os símbolos analisados. O protótipo inicia o funcionamento solicitando ao usuário a seleção de uma imagem, conforme pode-se ver na Figura 9. Para elaboração de testes e resultados foram utilizados cinco casos de teste.

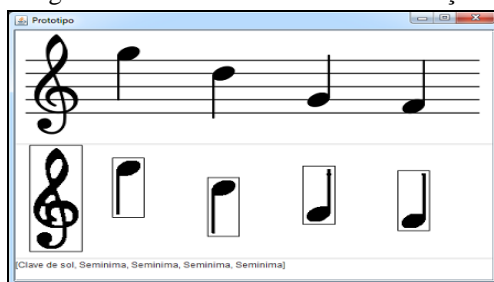
Figura 9 – Seleção de imagens



Fonte: Mauricenz (2013).

Após selecionar a imagem, inicia-se as etapas de remoção das linhas, segmentação e classificação. Ao término é apresentada uma tela para o usuário com o resultado da ação do protótipo sobre a imagem. Conforme Figura 10, na parte superior é mostrada a partitura utilizada para processamento.

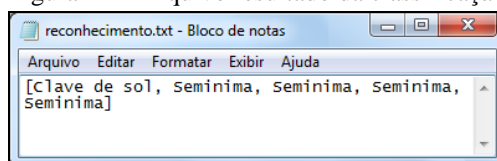
Figura 10 – Tela resultado da classificação



Fonte: Mauricenz (2013).

Além de visualizar o protótipo salva o resultado do processamento no mesmo diretório onde a imagem foi selecionada, na forma de um arquivo puro texto. Tal arquivo conterá entre colchetes e separados por vírgula o resultado do reconhecimento da partitura. Tal arquivo é mostrado Figura 11.

Figura 11 – Arquivo resultado da classificação



Fonte: Mauricenz (2013).

Segundo Mauricenz (2013, p. 55), devido à complexidade em relação a diversidade de possibilidades de apresentação de uma partitura, o protótipo limitou-se a partituras simples, sintéticas e sem arranjos. Quanto aos resultados e limitações, foi possível perceber que o algoritmo danificou parcialmente o símbolo durante a etapa de



remoção das linhas, porém esta falha era compensada dada a possibilidade posterior de isolar os símbolos individualmente e facilitar a etapa de reconhecimento. Ainda de acordo com o autor, a aplicação de descritores de Fourier para identificação da forma do símbolo musical mostrou-se eficaz. No entanto sugeriu mudanças para utilização de outras técnicas envolvendo aprendizado por máquina. Dentre as principais sugestões de melhorias estão: necessidade de remoção das linhas da pauta permitindo imagens em posição não horizontal, melhoria na abrangência do método de classificação para permitir outros símbolos musicais mais complexos, adicionar semântica para reconstrução dos símbolos musicais, permitir entrada de partituras impressas com ruídos. Sugeriu ainda permitir exportar o conteúdo interpretado para arquivos de notação musical como MUSICXML, uma vez que deu a entender que sua intenção é possibilitar inclusive exportar para formatos em que usuários com deficiências visuais possam compreender.

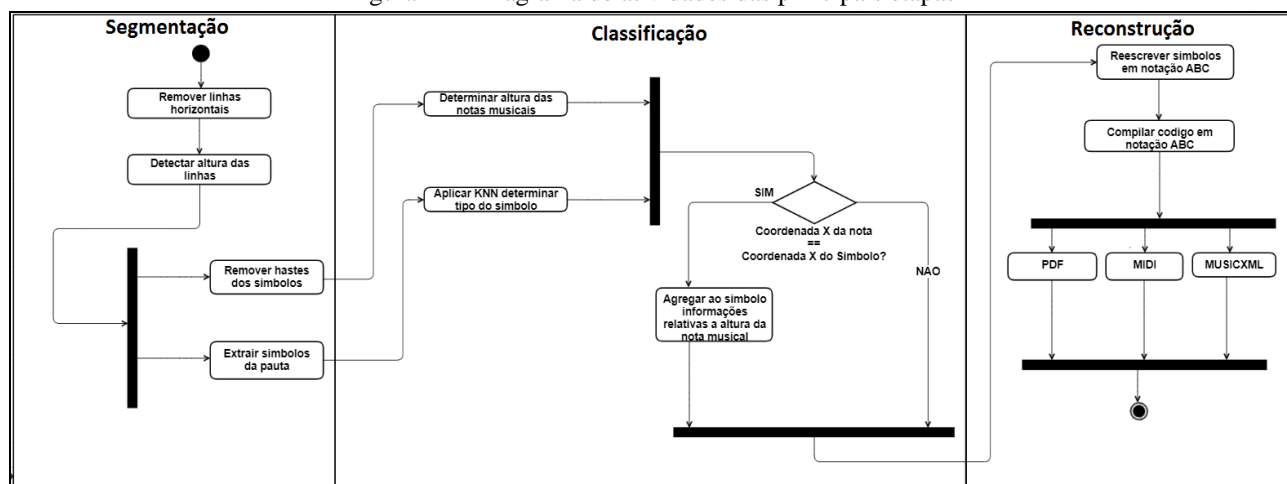
### 3 PROTOTIPO DESENVOLVIDO

Esta seção apresenta o desenvolvimento da extensão do protótipo Interpres. A Subseção 3.1 mostra uma visão geral do protótipo e suas principais funcionalidades. A subseção 3.2 trata dos detalhes de implementação do protótipo quanto cada uma de suas principais funcionalidades.

#### 3.1 VISÃO GERAL DO PROTÓTIPO

O protótipo tem como objetivo auxiliar o usuário na transcrição de uma partitura em imagem impressa para um formato em que seja possível editá-la posteriormente através de um software de partituras. É disponibilizado para usuário a possibilidade de exportar a partitura para formatos PDF, MIDI e MUSICXML. A Figura 12 apresenta o diagrama de atividades executadas pelo desde a segmentação até a reconstrução final mediante solicitação do usuário para processamento da imagem de partituras.

Figura 12 – Diagrama de atividades das principais etapas



Fonte: elaborado pelo autor.

A primeira tela do protótipo que é apresentada na Figura 13 dispõe ao usuário o campo *Arquivo* que possui ao seu lado direito um botão com três pontos. Ao clicar neste botão, será requisitada a localização da imagem de partitura no computador. Ao selecionar a imagem em seu diretório, esta será imediatamente carregada ao lado direito no painel de visualização *Imagem original*. Com a imagem carregada o usuário tem a opção de clicar no botão *Processar Conteúdo Partitura*. Ao fazê-lo, o protótipo iniciará a análise e o processamento da imagem. O Quadro 5 mostra a descrição das funções de cada componente.

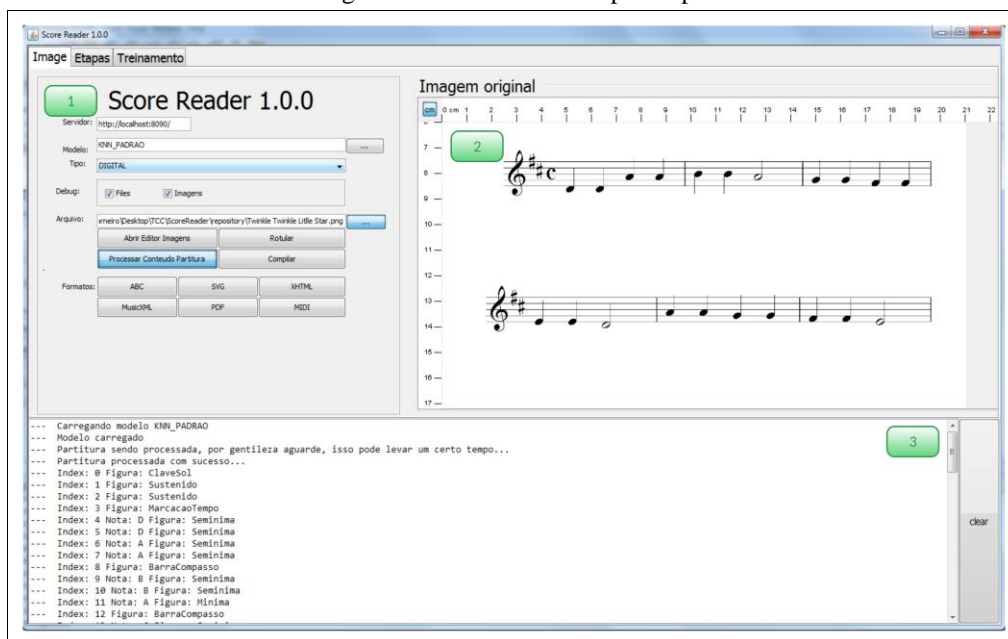
Quadro 5 – Descrição dos componentes da tela inicial

Identificador	Descrição
1	Painel de funcionalidades.
2	Painel de visualização de imagens.
3	Console de mensagens e erros.

Fonte: elaborado pelo autor.

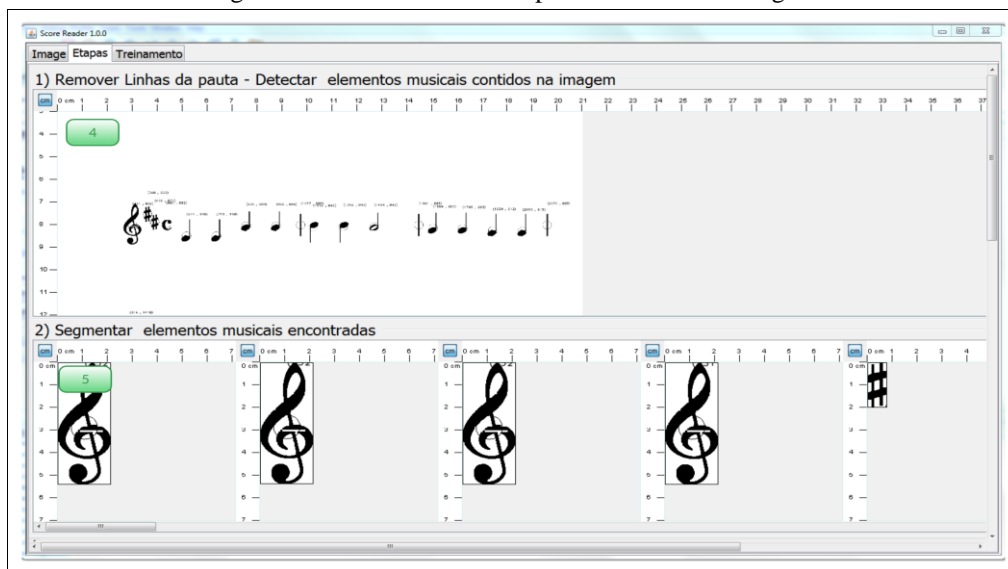
Ao finalizar o processamento da imagem da partitura, o usuário poderá acompanhar o resultado no console de mensagens e erros. Se desejar mais informações, na aba *Etapas* é possível ainda visualizar o resultado da fase segmentação da imagem. As Figuras 14 e 15 e 16 mostram os painéis detalhando estas fases. Em seguida, o Quadro 6 mostra a descrição de funcionalidade de cada painel.

Figura 13 – Tela inicial do protótipo



Fonte: elaborado pelo autor.

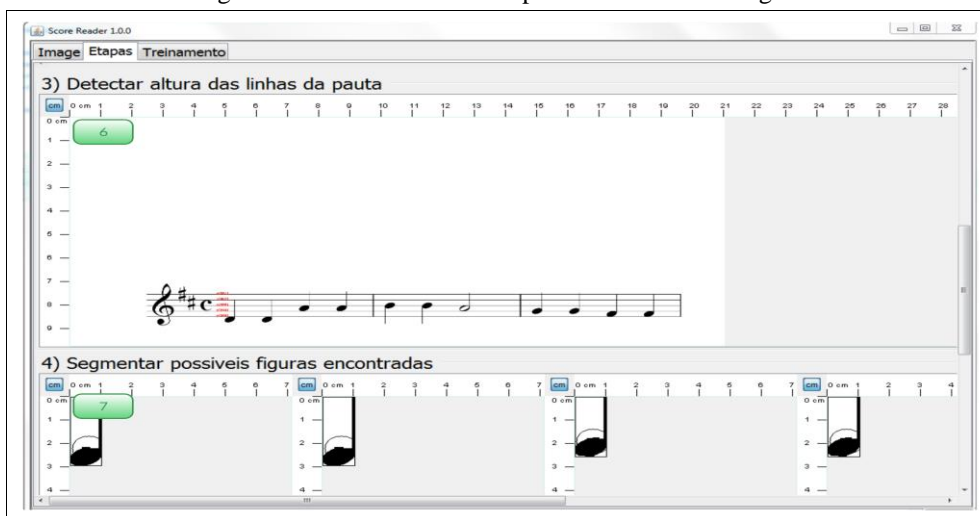
Figura 14 – Painéis de fase de processamento da imagem



Fonte: elaborado pelo autor.

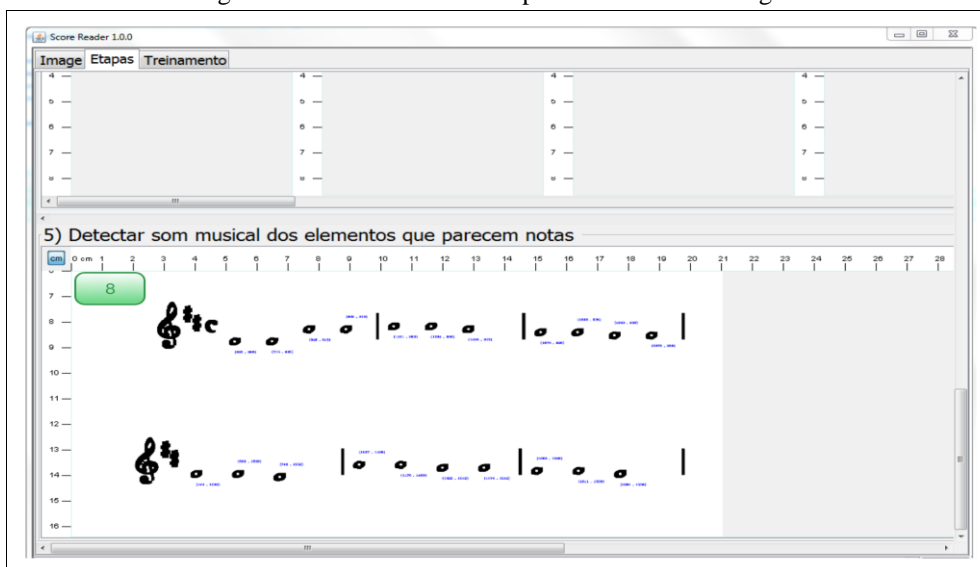


Figura 15 – Painéis de fase de processamento da imagem



Fonte: elaborado pelo autor.

Figura 16 – Painéis de fase de processamento da imagem



Fonte: elaborado pelo autor.

Quadro 6 – Descrição dos painéis

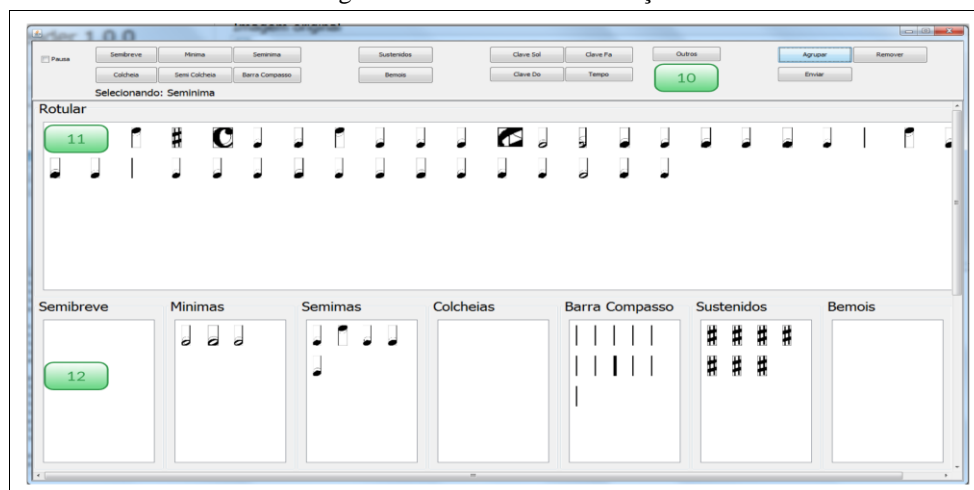
Identificador	Descrição
4	Resultado da ação do processo e remoção das linhas da imagem.
5	Resultado da segmentação dos símbolos da imagem.
6	Resultado da detecção da altura das linhas da pauta.
7	Resultado dos símbolos classificados como notas musicais.
8	Resultado da etapa de remoção das hastes e detecção da altura dos sons musicais em relação ao centro das elipses (ponto branco).

Fonte: elaborado pelo autor

O protótipo conta com um conjunto de dados próprio já treinado para reconhecimento e classificação de símbolos musicais. Porém caso o usuário preferir, tem a liberdade de criar o seu próprio conjunto de treino para utilizar em futura análise e reconstrução de partituras. Este conjunto de treino é elaborado mediante prévia rotulação.

Tirando proveito do algoritmo de segmentação de símbolos individuais do protótipo, é possível rotular imagens segmentadas. Esta funcionalidade está disponível através do botão *Rotular*, do painel de funcionalidades da tela principal. Uma vez que as imagens estiverem devidamente rotuladas, o usuário deve clicar no botão *Enviar* para que o novo conjunto rotulado seja adicionado ao modelo. A Figura 17 mostra a interface para rotulação das imagens. Em seguida, o Quadro 7 mostra a descrição das principais funcionalidades da tela de rotulação.

Figura 17 – Interface de rotulação



Fonte: elaborado pelo autor.

Quadro 7 – Descrição dos painéis

Identificador	Descrição
10	Painel de botões com os tipos de rótulos disponíveis.
11	Painel de símbolos musicais segmentados e pendentos de rotulação.
12	Caixas de agrupamento ou de rotulação.

Fonte: elaborado pelo autor.

Caso o usuário opte por rotular novas imagens e criar seu próprio conjunto de treinamento, é necessário dirigir-se à aba **Treinamento** e clicar no botão **Trein. Padrao**. Com esta ação de fato o treinamento será realizado. O protótipo ainda oferece nesta mesma tela recurso para importar um conjunto de dados de treinamento externo. O conjunto de dados compatível é o do MUSCIMA++ (HAJIC JUNIOR; PECINA, 2017), que contém imagens de símbolos manuscritos. Para a correta utilização do conjunto de treinamento externo, é ainda necessário selecionar na caixa de seleção ao lado do campo **DataSource** a opção **MANUSCRITO**, enquanto para utilizar o conjunto de dados do protótipo o usuário deve selecionar a opção **DIGITAL**. A Figura 18 mostra os componentes principais da tela de treino, que estão descritos no Quadro 8.

Quadro 8 – Descrição dos principais componentes

Identificador	Descrição
13	Painel de funcionalidades de treino
14	Gráfico de distribuição do modelo

Fonte: elaborado pelo autor

Figura 18 – Interface de treino



Fonte: elaborado pelo autor.

## 3.2 IMPLEMENTAÇÃO

O protótipo foi desenvolvido utilizando o conceito de aplicações cliente-servidor. Para desenvolvimento da parte cliente foi utilizada a linguagem de programação Java em conjunto com o framework Java Swing. Já na parte servidor foi utilizada a linguagem de programação Python, dada a sua simplicidade e facilidade de trabalhar com algoritmos de inteligência artificial. O motivo pelo qual optou-se em utilizar Java na parte cliente está diretamente ligado ao reaproveitamento de algumas funcionalidades do protótipo estendido.

Foi utilizada a versão 2.7 do Python por conta de algumas incompatibilidades encontradas no decorrer do desenvolvimento ao juntar outras bibliotecas. Tanto em Java quanto em Python foi utilizada a biblioteca OpenCV para processamento de imagens, tendo em vista a boa documentação e exemplos existentes na internet para estas duas linguagens.

No Apêndice A são apresentadas as principais classes e a forma como ambas estão estruturadas e relacionadas. As classes representam de forma genérica uma pauta musical e os elementos contidos nela. Estes elementos são *Figura* e *Nota*. Uma figura em algumas situações pode representar o som de uma nota musical, em outras somente um sinalizador. O diagrama ainda apresenta uma especialização da classe *Pauta*, no caso, *ClaveSol*. No Apêndice B são apresentadas as principais classes e principais métodos implementados na parte servidor do protótipo e o Apêndice C mostra as principais funcionalidades oferecidas pelo protótipo na forma de casos de uso

### 3.2.1 SEGMENTAÇÃO

O processo de segmentação é iniciado quando o usuário clica no botão *Processar* conteúdo disponível na tela principal do protótipo. A imagem é então encaminhada para o servidor, que faz a remoção das linhas horizontais. O Quadro 9 mostra o código referente a obtenção das coordenadas e remoção das linhas. A remoção é feita na linha 11 pelo método `remove_staves` da biblioteca *Musicstaves*. Além da remoção das linhas, o algoritmo determina a coordenada de cada linha e armazena em uma lista para utilização em fase posterior na linha 31. Finalizada a execução, a imagem manipulada é então é devolvida para o lado cliente do protótipo na linha 37. A remoção das linhas da pauta tem como objetivo facilitar a segmentação e o reconhecimento individual dos símbolos.

Quadro 9 – Remoção das linhas da pauta

```
1 def obterInformacoesPautas(self, imageEncoded):
2     retorno = []
3
4     #converte imagem base64 para imagem do Python
5     pilImage = self.convertToPILImage(imageEncoded)
6     gameraImage = self.convertToGameraImage(pilImage)
7
8     #algoritmo do musicstaves escolhido para remocao das linhas
9     ms = musicstaves_rl_simple.MusicStaves_rl_simple(gameraImage)
10    #remove as linhas
11    ms.remove_staves(crossing_symbols='bars', num_lines=5)
12
13    #obtem as pautas das imagens
14    pautas = ms.get_staffpos()
15
16
17    #salvar imagem para fins de debug
18    cv2.imwrite(self.DIR + 'posicaoLinhas.png', self.convertToCvImage(imageEncoded))
19    img = cv2.imread(self.DIR + 'posicaoLinhas.png')
20
21
22    for pauta in pautas:
23        linhas = []
24        for index, y in enumerate(pauta.yposlist):
25            #escreve a posicao das linhas para fins de debug
26            linhas.append({"linha": {"index": (index + 1), "y": y}})
27            #desenha informação na pauta para debug
28            cv2.putText(img, "y(" + str(y) + ")", (520, int(y)), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (0, 0, 255), 1, cv2.LINE_AA)
29
30            #adiciona coordenadas Y das linhas na lista
31            retorno.append({"pauta": {"index": pauta.staffno, "linhas": linhas, "yposlist": pauta.yposlist}})
32
33    #salvar imagem para fins de debug
34    cv2.imwrite(self.DIR + 'posicaoLinhas.png', img)
35
36    #retorna Json no formato string
37    return json.dumps(retorno)
```

Fonte: elaborado pelo autor.

Ao receber a imagem sem as linhas horizontais, ela é imediatamente convertida para escala de cinza. Esta conversão possibilita a aplicação de filtros de erosão e dilatação. Tais filtros têm como objetivo otimizar a qualidade da imagem. Esta otimização é necessária para o sucesso da etapa de detecção de formas do método `cvFindContours` do OpenCV, garantindo a correta forma dos símbolos e evitando que sejam removidas por ele partes importantes de cada símbolo. Uma vez segmentadas, as formas identificadas são adicionadas em uma lista e ordenadas da esquerda para direita e de cima para baixo. Esta ordenação tem como objetivo manter a ordem de leitura dos símbolos e para que estes

sejam reescritos na ordem em que estão na partitura. O Quadro 10 mostra os passos para segmentação dos elementos realizadas no método `cropElements`. Na linha 143 é possível visualizar o método `cvFindContours` responsável por identificar as formas conexas presentes na imagem.

Quadro 10 – Segmentação das figuras musicais

```

134  */
135  public static ArrayList<Crop> cropElements(opencv_core.IplImage imagemCinza, opencv_core.IplImage imagemBB) {
136
137      ArrayList<Crop> elements = new ArrayList<Crop>();
138
139      opencv_core.CvMemStorage storage = opencv_core.CvMemStorage.create();
140      opencv_core.CvSeq contours = new opencv_core.CvContour();
141
142      //Busca formas conexas na imagem
143      cvFindContours(imagemCinza, storage, contours, Loader.sizeof(opencv_core.CvContour.class),
144                  CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE, new opencv_core.CvPoint(0, 0));
145
146      List<opencv_core.CvSeq> contourList = new ArrayList<>();
147      for (; contours != null; contours = contours.h_next()) {
148          contourList.add(contours);
149      }
150
151      //Ordena da Direita para esquerda e de cima para baixo
152      contourList = sortLeftToRightTopToBottom(contourList);

```

Fonte: elaborado pelo autor.

### 3.2.2 CLASSIFICAÇÃO

Com a lista de elementos ordenados, aplica-se a etapa de identificação dos símbolos que representam sons musicais, a qual tem como objetivo identificar quais figuras são de fato notas musicais. A etapa avalia as coordenadas de cada figura para determinar inicialmente a qual das pautas o símbolo pertence. O Quadro 11 além de outras informações mostra a implementação do método que determina quais figuras pertencem a uma instância do objeto `Pauta`. Uma vez identificada a pauta que o símbolo pertence, a Figura é adicionada imediatamente sob responsabilidade do objeto `Pauta` conforme pode ser visualizado na linha 11.

Quadro 11 – Segmentação das figuras musicais

```

1  //Etapa obtém informacoes relativas a pauta, Linhas, Posicao das linhas etc
2  pautas = Utilities.obterInformacoesPautas(originalImage);
3  //Determina o meio dos espacos da pauta
4  for (Pauta clave : pautas) {
5      clave.calcularEspacos(); //Processa espacos
6  }
7  //Adiciona as figuras que pertencem a pauta
8  for (Pauta pauta : pautas) {
9      for (Figura figura : figuras) {
10         if (pauta.figuraDentroDaPauta(figura)) {
11             pauta.addFiguraPauta(figura);
12         }
13     }
14 }
15 ArrayList<Nota> notas = Utilities.detectarAlturaNotas(imagemSemLinhas);
16 for (Pauta pauta : pautas) {
17     for (Nota nota : notas) {
18         if (pauta.notaDentroDaPauta(nota)) {
19             pauta.addNotaPauta(nota);
20         }
21     }
22 }
23 //chama metodo da clave para determinar notas pertencentes a ela
24 for (Pauta clave : pautas) {
25     clave.determinarNotasPauta();
26 }
27 //chama metodo da clave para determinar altura das figuras que sao notas
28 for (Pauta clave : pautas) {
29     clave.determinarAlturaNotasPauta();
30 }
31 for (Pauta clave : pautas) {
32     clave.classificarFiguras(K.getText());
33 }

```

Fonte: elaborado pelo autor.

Uma vez identificado quais símbolos pertencem a cada objeto `Pauta`, o método `determinarNotasPauta` na linha 29, irá avaliar se o símbolo está sobre uma linha ou sobre um espaço. Uma observação a se fazer é que o algoritmo

de remoção das linhas da biblioteca Musicstaves fornece somente as coordenadas de altura das linhas removidas, portanto para determinar os espaços em branco existentes em uma partitura, é necessário um cálculo à parte. Este cálculo consiste em encontrar a posição central de cada espaço através da diferença das alturas das linhas imediatamente acima e abaixo. O Quadro 12 mostra a implementação do método `calcularEspacos` na linha 41. Ao observar o código especificamente na linha 56, é possível perceber que uma nova linha é adicionada entre as linhas existentes que representarão de fato os espaços entre estas.

Quadro 12 – Cálculo dos espaços entre as linhas

```

40
41 public void calcularEspacos() {
42     for (int i = 1; i <= getLinhas().size(); i++) {
43
44         Linha l = getLinhas().get(String.valueOf(i));
45         Linha lb = getLinhas().get(String.valueOf(i + 1));
46         Linha la = getLinhas().get(String.valueOf(i - 1));
47
48         if (lb != null) {
49             double diff = Math.abs((l.y - lb.y) / 2);
50             l.yEspacoAbaixo = l.y + diff;
51         }
52         if (la != null) {
53             double diff = Math.abs((l.y - la.y) / 2);
54             l.yEspacoAcima = l.y - diff;
55         }
56         getLinhas().put(String.valueOf(i), l);
57     }
58 }
59

```

Fonte: elaborado pelo autor.

Para determinar a altura de uma nota musical em relação as linhas da pauta, ainda foram aplicados filtros de erosão e dilatação sobre imagem da partitura, de tal forma que as hastes contidas nestas figuras são eliminadas, restando somente a forma elíptica do símbolo. Uma vez eliminadas as hastes, aplica-se também o algoritmo `cvFindContours`, porém desta vez para identificar tais formas elípticas e determinar o centro de cada uma delas. Com o centro determinado, é possível avaliar por proximidade se a figura está sobre uma linha ou sobre um espaço da pauta, e por consequência determinar também seu nome e som. O Quadro 13 mostra a implementação do método `determinarAlturaNotasPauta`, responsável por determinar o nome da nota musical por proximidade.

Quadro 13 – Determinar altura das notas musicais

```

95 public void determinarAlturaNotasPauta() {
96     for (Figura figura : figurasPauta) {
97         if (figura.isNotaMusical()) {
98             for (int i = 1; i <= getLinhas().size(); i++) {
99
100                 Linha linha = getLinha(String.valueOf(i));
101                 int diff = ((int) (figura.nota.y - linha.y));
102                 int variacao = Math.abs(diff);
103                 if (variacao < fator) { //Diferença for de menos de 2 pixels
104                     figura.nota.nome = getNomeNota(linha.index);
105                 } else {
106                     diff = ((int) (figura.nota.y - linha.yEspacoAbaixo));
107                     variacao = Math.abs(diff);
108                     if (variacao < fator) {
109                         //Pegar nota acima da pauta
110                         figura.nota.nome = getNomeNotaAbaixo(linha.index);
111                     } else {
112                         diff = ((int) (figura.nota.y - linha.yEspacoAcima));
113                         variacao = Math.abs(diff);
114                         //Pegar nota acima da pauta
115                         if (variacao < fator) {
116                             figura.nota.nome = getNomeNotaAcima(linha.index);
117                         }
118                     }
119                 }
120             }
121         }
122     }
123 }

```

Fonte: elaborado pelo autor.

Com os símbolos todos já identificados como sendo nota ou não, o protótipo avança para a etapa de reconhecimento dos símbolos. Esta etapa tem como objetivo determinar a duração caso seja uma nota, ou se for qualquer outro símbolo, determinar a ação que ele exercerá sobre a nota ou sobre pauta. Este processo de classificação é executado no servidor. Foi utilizada a implementação do algoritmo KNN em Python, disponibilizado pela biblioteca `scikit-learn`. O processo de classificação ocorre enviando individualmente cada figura ao servidor. Ao receber a figura o

servidor irá normalizá-la deixando suas características semelhantes às das notas utilizadas para treino. A imagem, portanto, será redimensionada, binarizada e por fim transformada em uma matriz unidimensional (vetor). Para transformá-la destaca-se o uso do método `flatten` do módulo `numpy` e que pode ser visualizado na linha 6. O Quadro 14 mostra a implementação do método `classificar`. Dependendo do resultado do método `predict` do KNN que pode ser visualizado na linha 12, o algoritmo irá retornar o índice que representa a classe a que o símbolo pertence.

Quadro 14 – Classificação dos símbolos

```

1 def classificar(self, img, K):
2     #redimensiona imagem para tamanho padrao
3     img = img.resize((self.LARGURA, self.ALTURA))
4     cinza = img.convert('L') # converte para escala cinza
5     cinza = cinza.point(lambda x: 0 if (x < 128) else 255, '1') # binarizacao da imagem [0,0,1,0,0,1,0, ...]
6     elemento_teste = numpy.array(cinza, dtype='uint8').flatten() # converte em array e achata imagem transformando ela em array de caracteristicas
7
8     #classes disponiveis
9     classes = ['Seminima', 'Minima', 'ClaveDo', 'ClaveSol', 'ClaveFa', 'PausaColcheia', 'PausaSeminima',
10              'PausaSemiColcheia', 'PausaSemiBreve', 'PausaMinima', 'BarraCompasso',
11              'Sustenido', 'Bemol', 'MarcacaoTempo', 'Colcheia', 'Semibreve', 'OutrosSimbolos']
12     encontrado = self.KNN.predict([elemento_teste]) #KNN prediz padrao de acordo com as caracteristicas
13     return classes[encontrado[0]] #retorna o nome relativo a classe

```

Fonte: elaborado pelo autor.

### 3.2.3 TREINAMENTO DO CLASSIFICADOR

Conforme já explanado, para o reconhecimento de padrões de símbolos, é possível fazer a rotulação dentro do próprio protótipo ou importando dados rotulados pelo MUSCIMA++. As figuras rotuladas no protótipo são armazenadas em um objeto da classe `ScoreReaderDataSet` e salvas em disco para futura recuperação. O Quadro 15 mostra um pequeno trecho da implementação do método `treinar`.

Este método recebe como parâmetro uma lista de figuras que são normalizadas nas linhas 6 e 7, rotuladas nas linhas 10 e 11 e convertidas em um vetor unidimensional na linha 17. Esta conversão da imagem para um vetor unidimensional ocorre para que cada bit da imagem funcione como uma representação das características da imagem. Para treinamento é fornecido ao método `fit` do `KNeighborsClassifier` na linha 28, um conjunto de treino obtido através do método `train_test_split` na linha 21. O método `train_test_split` é responsável por dividir a lista de figuras já rotuladas em um conjunto de teste e treino. O parâmetro `test_size` determina a proporção do conjunto de teste que deverá ser utilizado para validação do KNN. O método `fit` é o principal responsável por ajustar ou calcular as distancias de acordo com o conjunto de treino. O parâmetro `n_neighbors` do método `KNeighborsClassifier` representa o número de vizinhos próximos que devem ser consultados para determinar a que classe determinado símbolo pertence. Como não é fornecido explicitamente o parâmetro `metric` ao método `KNeighborsClassifier`, ele utilizará o cálculo de área ou métrica Minkowsky para determinar as distancias entre os vizinhos no espaço de distribuição.

Quadro 15 – Treinamento KNN

```

1 def treinar(self, tipo, figuras):
2     # Este trecho de codigo representa um pequeno pedaço relevante do algoritmo de treino, outros detalhes
3     # foram omitidos por serem repeticao somente
4
5     #extrai imagens da lista do conjunto de dados e redimensiona para manter padrao de altura e largura
6     img_sembreves = [self.extrair_imagem(tipo, cc) for cc in semibreves]
7     img_outros_simbolos = [resize(mn, (self.ALTURA, self.LARGURA)) for mn in img_outros_simbolos]
8
9     #rotula as imagens contidas nas listas do conjunto
10    rotulos_sembreves = [self.ROTULO_SEMIBREVE for _ in img_sembreves]
11    rotulos_outros_simbolos = [self.ROTULO_OUTROS for _ in img_outros_simbolos]
12
13    #depois de rotulada torna a misturar imagens
14    self.misturadas = img_sembreves + img_outros_simbolos
15
16    # converte imagem em matrix unidimensional para transformar bits da imagem em caracteristicas
17    self.figuras_array_linha = [n.flatten() for n in self.misturadas]
18    self.rotulos_classe = rotulos_sembreves + rotulos_outros_simbolos
19
20    #separa o conjunto de testes e treino
21    self.X_conjunto_treino, self.X_conjunto_teste, self.Y_conjunto_treino, self.Y_conjunto_teste = train_test_split(
22        self.figuras_array_linha, self.rotulos_classe, test_size=0.40, random_state=42,
23        stratify=self.rotulos_classe)
24
25    #cria chamada para KNN
26    self.KNN = KNeighborsClassifier(n_neighbors=self.K_VIZINHOS_PROXIMOS)
27    #utiliza para treino do KNN o conjunto X e Y de treinos separados
28    self.KNN.fit(self.X_conjunto_treino, self.Y_conjunto_treino)

```

Fonte: elaborado pelo autor.



### 3.2.4 RECONSTRUÇÃO EM LINGUAGEM NOTAÇÃO ABC

Com os símbolos todos classificados e devidamente reconhecidos, o protótipo pode avançar para a etapa de reconstrução da partitura em uma linguagem intermediária. Esta etapa consiste em representar todos os elementos em um arquivo com extensão `abc` utilizando a sintaxe e semântica exigida pela linguagem. O arquivo gerado conterá informações relativas a cabeçalho e logo abaixo, as figuras musicais que aparecem na partitura. O Quadro 16 mostra a implementação do método `getElementos` responsável por navegar sobre as pautas e transcrever os símbolos para a linguagem intermediária.

Quadro 16 – Extração dos elementos

```
1 private String getElementos(ArrayList<Pauta> pautas) {
2     StringBuilder abcCode = new StringBuilder();
3     int qtQuebra = 0;
4     for (Pauta pauta : pautas) {
5         boolean inicioNotasFimClave = false;
6         String armaduraClave = getElementoArmaduraClave(pauta);
7         abcCode.append(armaduraClave);
8
9         for (Figura elemento : pauta.getFigurasPauta()) {
10            if (elemento.isNotaMusical()) {
11                abcCode.append(elemento.getNota().nome).append(getTipoNota(elemento));
12                inicioNotasFimClave = true;
13            } else if (inicioNotasFimClave) {
14
15                String tipo = getTipoPausa(elemento);
16                if (tipo.isEmpty()) {
17                    tipo = getTipoFigura(elemento);
18                }
19                abcCode.append(tipo);
20                qtQuebra++;
21            }
22        }
23    }
24    abcCode.append(ABC_NOTATION.FINAL);
25    return abcCode.toString();
26 }
27 }
```

Fonte: elaborado pelo autor.

Com o arquivo `abc` já gerado, o usuário pode através do botão `Compilar` disponível na tela inicial do protótipo, visualizar as saídas nos formatos PDF, MIDI e MUSICXML. Por meio deste botão os compiladores disponíveis para notação ABC são acionados dentro do protótipo através de arquivos com extensão `bat` tendo como entrada o arquivo em linguagem intermediária.

## 4 RESULTADOS

Este capítulo está dividido em três partes para apresentar os resultados obtidos com o desenvolvimento deste trabalho. A primeira parte apresenta os resultados em relação aos objetivos propostos, a segunda parte mostra a avaliação em comparação com trabalho estendido e a terceira parte apresenta a avaliação em comparação com os correlatos.

### 4.1 AVALIAÇÃO EM RELAÇÃO AOS OBJETIVOS PROPOSTOS

Para avaliar os resultados do trabalho em relação aos objetivos propostos, o classificador do protótipo precisou ser treinado com símbolos extraídos de diferentes imagens de partituras. Algumas foram elaboradas pelo autor através do aplicativo de edição de partituras MUESCORE (2019) e outras foram obtidas da internet. As partituras obtidas da internet são de duas músicas simples para instrumentos de uma única voz. O autor elaborou quatro escalas musicais contendo todos os diferentes tipos de símbolos suportados pelo protótipo. Os testes se limitaram a partituras mais simples, sem muitos arranjos e sem acordes e o classificador está limitado em dezoito tipos de símbolos musicais. Foram elaborados seis diferentes tipos de treinos. Isso foi necessário porque dependendo da quantidade de imagens e treinamento que o classificador recebia, ele identificava incorretamente alguns símbolos. A Figura 19 mostra um exemplo de imagem de partitura na qual os seus símbolos foram extraídos para treinamento.

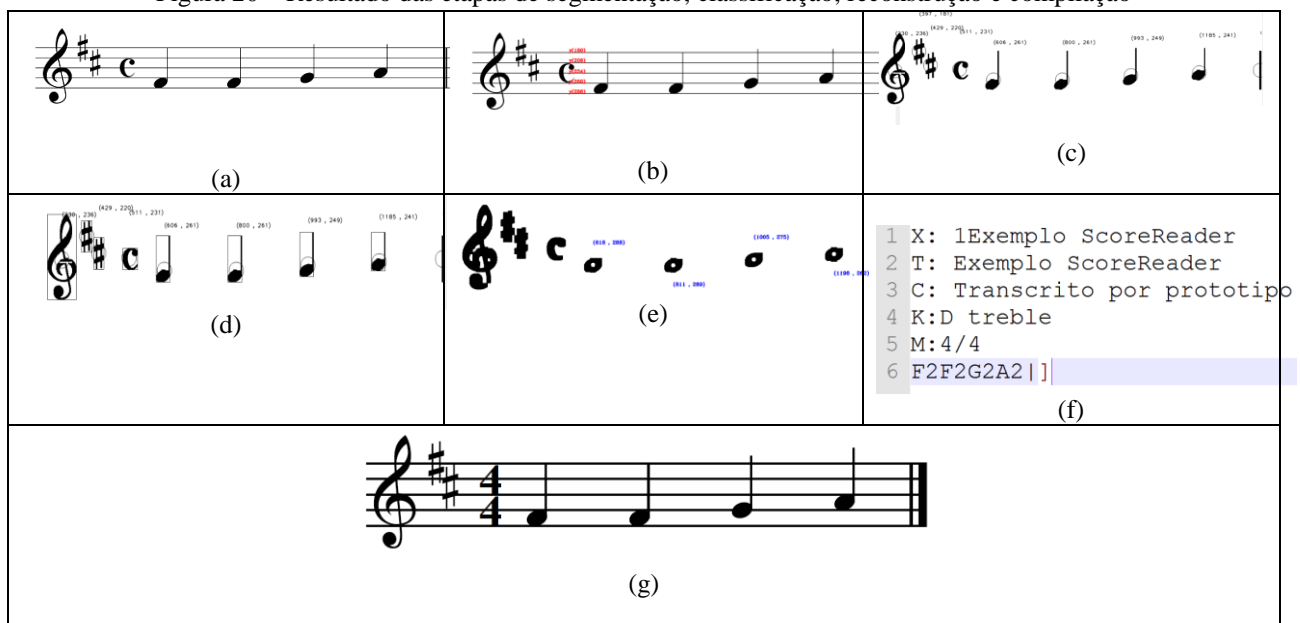
Figura 19 - Imagem utilizada de treinamento do classificador



Fonte: elaborado pelo autor.

Para realizar o reconhecimento e reconstrução da partitura, a imagem é tratada e seus símbolos são extraídos e encaminhados ao classificador e reescritos em linguagem intermediária. O resultado final é a partitura compilada pelo protótipo para os formatos PDF, MIDI e MUSICXML. Esta sequência de etapas é aplicada a todas as imagens de partituras processadas através do protótipo. A Figura 20 mostra a ação de cada uma das etapas de forma individual, iniciando pela etapa de carregamento da imagem (a), posteriormente detecção da altura das linhas e remoção das linhas horizontais através da biblioteca do Musicstaves (b)(c), segmentação dos símbolos individuais (d), remoção das hastes para determinar o centro das figuras elípticas e nome da nota, reescrita da partitura em linguagem intermediária (f) e por fim o resultado da compilação do código para formato PDF (g). Para geração do PDF em específico foi utilizada a biblioteca BATIK, visto que o compilador da *abcmnotation* para o formato PDF foi descontinuado.

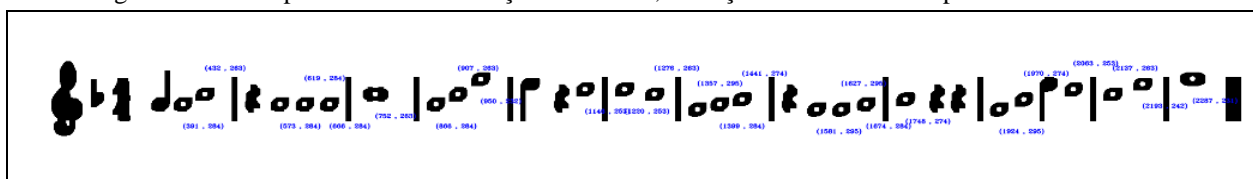
Figura 20 – Resultado das etapas de segmentação, classificação, reconstrução e compilação



Fonte: elaborado pelo autor.

Ao executar a sequência de etapas para outras imagens de partituras o protótipo teve dois tipos de resultados negativos. O primeiro ocorre na etapa que corresponde a segmentação dos símbolos individuais (d), pois algumas figuras podem ter seu corpo separado da haste dependendo da baixa nitidez da imagem. Provavelmente isso ocorre porque o número de interações de erosão e dilatação que é aplicado a imagem está fixo, e dependendo da imagem, um número maior ou menor de interações deveria ser aplicado. O segundo corresponde à etapa de detecção das formas elípticas e seu respectivo centro (e), pois também devido ao número de erosões e dilatações, a imagem pode não ter sua haste completamente removida e com isso não será possível determinar o centro da elipse. Isso faz com que o algoritmo que detecta a altura das notas falhe ao identificar o nome da nota em questão. Para resolver esta falha provavelmente deveria ser revisto o algoritmo para que, mesmo com as hastes seja possível determinar as elipses e seus respectivos centros. A Figura 21 mostra um exemplo de situação em que ocorreu falha em três notas da partitura. Nota-se que as figuras que foram devidamente identificadas como notas possuem um ponto branco ao centro.

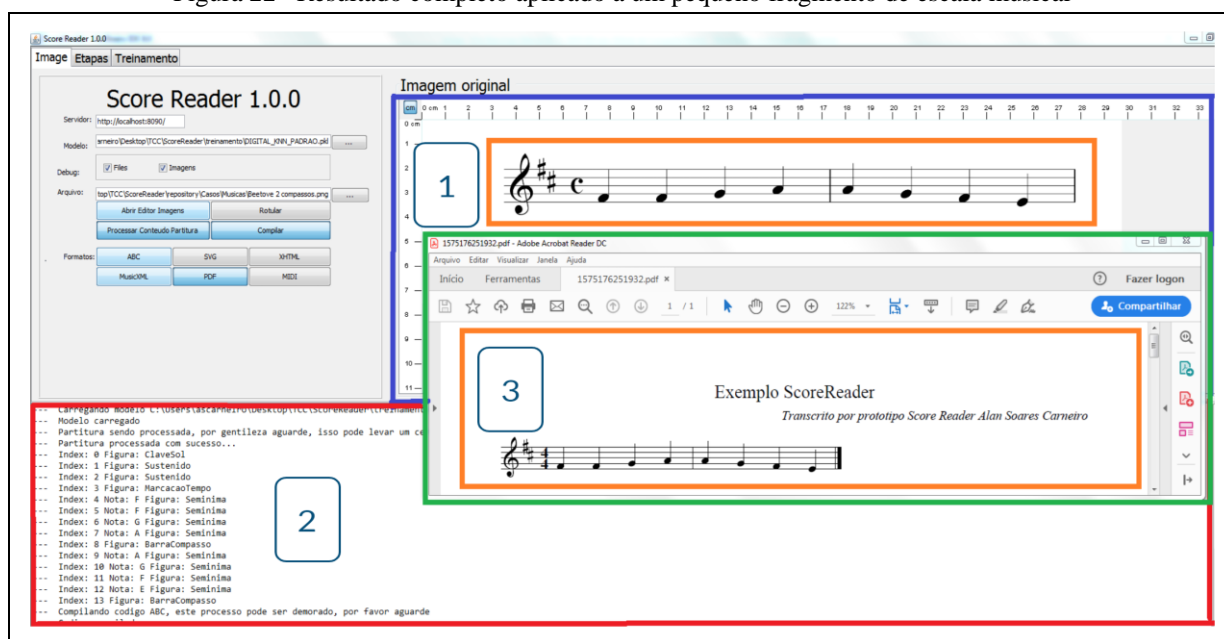
Figura 21 – Exemplo de falha da remoção das hastes, detecção do centro das elipses e nome das notas



Fonte: elaborado pelo autor.

A Figura 22 mostra um pequeno exemplo de caso completo de classificação bem sucedido aplicado a um escala musical e seu resultado na tela do protótipo. O Quadro 17 apresenta cada uma breve descrição dos componentes da tela rotulados na imagem.

Figura 22 –Resultado completo aplicado a um pequeno fragmento de escala musical



Fonte: elaborado pelo autor.

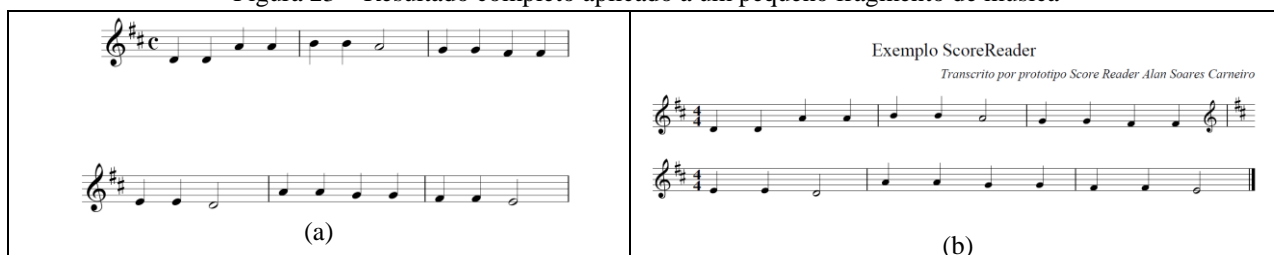
Quadro 17 – Descrição dos compiladores para MIDI, SVG e MUSICXML

Identificador	Descrição
1	Imagem de partitura carregada para o protótipo
2	Console apresentando informações sobre a classificação como: cifra (nome da nota) e tipo do símbolo reconhecido pelo classificador
3	Resultado final da conversão da partitura em formato PDF.

Fonte: elaborado pelo autor

Uma vez classificadas as figuras e detectadas as alturas das notas, a partitura pode ser reescrita utilizando a linguagem intermediária notação ABC. Para avaliar o sucesso da compilação das partituras reescritas para PDF, MIDI e MUSICXML. Foram efetuados testes com alguns exemplos de partituras, na Figura 23 é apresentado um caso envolvendo mais símbolos musicais através de um pequeno trecho musical. À esquerda é mostrada a música em sua imagem original (a) e à direita o resultado da geração de um PDF (b).

Figura 23 – Resultado completo aplicado a um pequeno fragmento de música

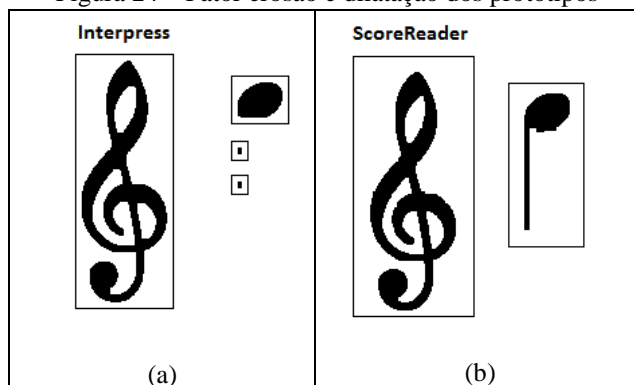


Fonte: elaborado pelo autor.

## 4.2 AVALIAÇÃO EM RELAÇÃO AO PROTÓTIPO ESTENDIDO

Para realizar a extração dos símbolos individuais contidos na partitura, optou-se por reaproveitar a implementação já existente no trabalho estendido (MAURICENZ, 2013). Entretanto o fator de erosão e dilatação utilizado precisou sofrer um pequeno ajuste na quantidade de iterações, pois ele separava a cabeça da haste ou em alguns casos eliminava a haste transformando em outras formas menores. Com isso o classificador passava a receber as informações separadas, causando uma falha na classificação da figura. A Figura 24 mostra um comparativo da segmentação de dois símbolos por ambos os protótipos. Logo à esquerda é possível ver o resultado da erosão realizada no protótipo estendido (a) e à direita (b) no novo protótipo com o número de erosões reformulado.

Figura 24 – Fator erosão e dilatação dos protótipos



Fonte: elaborado pelo autor.

Para avaliar a precisão do protótipo atual no reconhecimento de partituras em comparação com o protótipo estendido, foram submetidos os quatro primeiros experimentos utilizados no Interpres. A Tabela 1 mostra os resultados da análise destes experimentos. A coluna “Símbolos” representa a quantidade de símbolos musicais contidos em cada imagem de partitura, a coluna “ScoreReader (2019)” representa a quantidade de símbolos reconhecidos no protótipo atual e a coluna “Interpres (2013)” representa a quantidade de símbolos reconhecidos no protótipo que foi estendido.

Tabela 1 – Análise do e comparativo do resultado dos quatro primeiros experimentos

Testes	Símbolos	ScoreReader (2019)	Interpres (2013)
Experimento 1	5	5	5
Experimento 2	8	8	8
Experimento 3	4	4	-
Experimento 4	20	20	17
TOTAL	37	37	30

Fonte: elaborado pelo autor.

Através da análise das colunas “Símbolos” e “ScoreReader (2019)” é possível concluir que a taxa de acerto para o reconhecimento dos símbolos no protótipo atual foi de 100%, apresentando pequena evolução. De todos os quatro experimentos reavaliados no protótipo “Interpres (2013)”, ele mostrou dificuldades no reconhecimento de alguns símbolos do “Experimento 3” e “Experimento 4”, mas que segundo Mauricenz (2013, p. 55), ocorre quando os símbolos estão de alguma forma unidos. No apêndice C Quadro 28 é apresentado o resultado da saída do console para cada um dos quatro experimentos. Os protótipos foram submetidos a dois novos experimentos contendo duas folhas de música com total de 62 símbolos, dos quais ambos os protótipos estão preparados para reconhecer. A Tabela 2 mostra o resultado. Para elaboração do resultado foi necessário redimensionar a imagem dos trechos musicais utilizando como parâmetro o tamanho das imagens dos experimentos da Tabela 1 para então fazer a comparação. A coluna “Música” representa o nome do trecho musical utilizado como experimento, a coluna “Símbolos” a quantidade de símbolos presente em cada experimento, enquanto as colunas “ScoreReader (2019)” e Interpres (2013) mostram os respectivos resultados da classificação de cada um dos protótipos.

Tabela 2 – Trechos musicais

Música	Símbolos	ScoreReader (2019)	Interpres (2013)
Brilha brilha estrelinha	35	35	12
Ode Alegria	27	26	6
TOTAL	62	61	18

Fonte: elaborado pelo autor.

Conforme a mostra a Tabela 2, o protótipo Interpres não pôde determinar alguns símbolos corretamente nos dois experimentos, enquanto que o protótipo atual obteve melhor êxito em um contexto em que mais figuras e

elementos estão presentes. No Apêndice E são apresentados os dois novos experimentos e o respectivo resultado em cada protótipo. No Apêndice F é mostrado o resultado da segmentação na tela de ambos os protótipos, nele é possível notar que no “Interpress (2013)” alguns símbolos tiveram parte da estrutura removida por conta da segmentação fazendo com que o classificador mostre dificuldades no reconhecimento individual deste símbolo. Esta avaliação teve como objetivo comparar se houve evolução entre os protótipos.

#### 4.3 AVALIAÇÃO EM RELAÇÃO AOS CORRELATOS

No Quadro 18 é demonstrada a comparação das características entre o protótipo atual em relação aos trabalhos correlatos e também em relação ao protótipo anterior. Através dele é possível concluir que todos os protótipos têm em comum a fase de remoção das linhas da pauta como etapa essencial para a segmentação dos símbolos. É possível observar que somente os protótipos de Silva (2017) e Castro (2014) suportam o processamento de partituras para instrumento de mais de uma voz, o que demonstrou ser melhor neste aspecto. Com relação à versão anterior do protótipo “Mauricenz (2013)” e o protótipo atual “ScoreReader (2019)” pode-se observar uma série de evoluções em comparação com as funcionalidades dos correlatos.

Quadro 18 – Comparação de correlatos e protótipo anterior

Características \ Correlatos	Mauricenz (2013)	Castro (2014)	Jesus (2015)	Silva (2017)	ScoreReader (2019)
Faz uso da etapa de remoção das linhas da pauta	Sim	Sim	Sim	Sim	Sim
Reconstrução de partitura a partir de imagem	Não	Não	Não	Sim	Sim
Utiliza IA (Inteligência artificial) para classificação dos símbolos	Não	Sim	Não	Sim	Sim
Suportam partituras com instrumento de mais de uma voz	Não	Sim	Não	Sim	Não
Permite criar modelo de treinamento no protótipo	Não	Não	Não	Não	Sim
Permite utilizar conjunto de treino externo	Não	Não	Não	Sim	Sim
Utiliza linguagem intermediária para geração da partitura	Não	Não	Não	Sim	Sim
Exportação da partitura em PDF	Não	Não	Não	Não	Sim
Exportação da partitura em MIDI	Não	Não	Sim	Não	Sim
Exportação da partitura para MUSICXML	Não	Não	Não	Sim	Sim

Fonte: elaborada pelo autor.

Quanto à possibilidade de exportação para formato MIDI somente o trabalho de Jesus (2015) e ScoreReader (2019) apresentam esta funcionalidade, embora por permitir a geração de saída da partitura para o formato MUSICXML o protótipo de Silva (2017) também possa permitir isso futuramente. Quanto ao suporte para exportação em diferentes formatos, pode-se concluir que o protótipo ScoreReader (2019) contém uma gama maior de possibilidades. Por utilizar uma linguagem intermediária para reconstrução da partitura outros formatos ainda poderão ser acrescentados, desde que haja um compilador para tal formato. Pode-se concluir que de todas as funcionalidades listadas, o protótipo atual ScoreReader (2019) com exceção do suporte para partituras de instrumentos de mais de uma voz, contempla a grande maioria, mostrando uma pequena evolução em relação aos demais.

## 5 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um protótipo capaz de fazer o reconhecimento e reconstrução de partituras a partir de uma imagem sem ruídos, sem muitos arranjos musicais e para instrumentos de uma única voz, utilizando para isso a linguagem intermediária notação ABC, com suporte à compilação nos formatos PDF, MIDI e MUSICXML. Portanto todos os objetivos específicos propostos foram atendidos.

A extensão do protótipo anterior buscou evoluir algumas funcionalidades sugeridas pelo autor, possibilitando a classificação dos símbolos utilizando algoritmos de inteligência artificial e geração da partitura em uma linguagem intermediária para que outros formatos possam ser alcançados a partir da partitura reconstruída. É possível agora a compilação da notação musical reescrita para formatos MIDI e MUSICXML. O arquivo MIDI poderá ser ouvido e o arquivo MUSICXML poderá ser interpretado através de um editor que suporte geração de saídas em formato braile, contribuindo assim de alguma forma para que pessoas com deficiência visual possam ter acesso ao aprendizado musical - uma das essências do Interpress.

O processo de compilação musical oferecido possibilita não somente gerar exportação para os formatos principais propostos, mas também o permite para SVG e XHTML. Também dispõe de recurso para rotulação de imagens e criação de modelo de treino dentro do próprio protótipo bem como a importação de modelo de treino externo.

Ressaltam-se algumas limitações e melhorias necessárias no trabalho. Dentre elas está o quesito suporte a partituras com instrumentos de mais de uma voz, algumas falhas de reconhecimento nas etapas de segmentação dos símbolos e detecção nos nomes das notas decorrentes de necessidade de adaptação e melhoria na quantidade de iterações de erosão e dilatação aplicados a cada tipo de imagem.

Espera-se que, com o desenvolvimento e evolução deste protótipo ele possa auxiliar de alguma forma músicos que necessitem transcrever de forma mais ágil uma partitura para um formato digital e que os poupe de algum trabalho maçante e repetitivo na necessidade de reeditá-las. É desejado também que de alguma forma o novo protótipo permita ajudar na digitalização e preservação de documentos de partituras antigas. Como extensões sugere-se:

- 1) Utilizar outras técnicas para segmentação dos símbolos possivelmente a combinação de descritores de forma com um algoritmo de inteligência artificial;
- 2) Permitir a entrada de partituras com ruídos ou danificadas pelo tempo;
- 3) Adicionar suporte a novos símbolos, como ligaduras, fermatas, pontos de aumento dentre outros;
- 4) Permitir suporte a partituras escritas em notação musical antiga;
- 5) Suporte a partituras com instrumentos de mais de uma voz;
- 6) Possibilitar exportar o código compilado para impressão em formato braile.

## REFERÊNCIAS

- CASTRO, André Filipe Ferreira de. **Reconhecimento de símbolos musicais em imagens cinza de partituras manuscritas**. 2014. 58 f. Dissertação (Mestrado) - Curso de Mestrado Integrado em Engenharia Eletrotécnica e de Computadores Major Automação, Mestrado Integrado em Engenharia Eletrotécnica e de Computadores Major Automação, Faculdade de Engenharia da Universidade do Porto, Portugal, 2014. Disponível em: <[https://sigarra.up.pt/reitoria/pt/pub\\_geral.show\\_file?pi\\_doc\\_id=26921](https://sigarra.up.pt/reitoria/pt/pub_geral.show_file?pi_doc_id=26921)>. Acesso em: 17 nov. 2019.
- CUNNINGHAM, Padraig; DELANY, Sarah Jane. K-Nearest Neighbour Classifiers. **Mult classif syst**. Dublin, p. 1-18. 27 abr. 2007. Disponível em: <[https://www.researchgate.net/publication/228686398\\_k-Nearest\\_neighbour\\_classifiers](https://www.researchgate.net/publication/228686398_k-Nearest_neighbour_classifiers)>. Acesso em: 18 nov. 2019.
- DALITZ, Christoph et al. **Musicstaves toolkit**. 2008. Disponível em: <<https://gamera.informatik.hsnr.de/addons/musicstaves/>>. Acesso em: 17 nov. 2019.
- DALITZ, Christoph; DROETTBOOM, Michael; FUJINAGA, Ichiro. **Gamera Project**. 2002. Disponível em: <<https://gamera.informatik.hsnr.de/>>. Acesso em: 17 nov. 2019.
- GONZATO, Guido. **Uso de ABC para escrever Música**. 2003. Disponível em: <<http://alfarrabio.di.uminho.pt/~albie/abc/pt-abc-manual-1.0.3-beta.pdf>>. Acesso em: 18 nov. 2019.
- HAJIC JUNIOR, Jan; PECINA, Pavel. MUSCIMA++. **Lindat/clarin**, Czech Republic, v. 1, n. 1, p.3704-3709, 25 ago. 2017. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Disponível em: <[https://ufal.mff.cuni.cz/~hajicj/2017/docs/icdar2017\\_CAMERA.pdf](https://ufal.mff.cuni.cz/~hajicj/2017/docs/icdar2017_CAMERA.pdf)>. Acesso em: 02 dez. 2019.
- INTEL. **OpenCV**: Open Source Computer Vision Library. 2019. Disponível em: <<https://opencv.org/about/>>. Acesso em: 17 nov. 2019.
- JESUS, Carlos Eduardo Zati de. **Sistema autônomo de omr para partituras**. 2015. 23 f. TCC (Graduação) - Curso de Engenharia da Computação, Universidade Positivo, Curitiba, 2015. Disponível em: <<https://www.up.edu.br/blogs/engenharia-da-computacao/wp-content/uploads/sites/6/2015/12/2015.Carlos.pdf>>. Acesso em: 17 nov. 2019.
- LACERDA, Hudson. **Introdução à notação ABC**. 2009. Disponível em: <<https://hudlac.files.wordpress.com/2009/11/introducao.pdf>>. Acesso em: 18 nov. 2019.
- MAILLARD, Philippe. **Introdução ao processamento digital de imagens**. Belo Horizonte: Vídeo, 2001. 30 slides, color. Disponível em: <<https://web.archive.org/web/20171118115920/http://www.csr.ufmg.br/geoprocessamento/publicacoes/cursopdi.pdf>>. Acesso em: 18 nov. 2019.
- MARGARIDA, Thiago. **Reconhecimento de imagens aplicado a partituras musicais**. 2009. 83 f. TCC (Graduação) - Curso de Bacharelado em Ciência da Computação, Universidade do Estado de Santa Catarina, Joinville, 2009.
- MARIZ, Filipe mendes. **Avaliação e comparações de versões modificadas do algoritmo knn**. 2017. 54 f. TCC (Graduação) - Curso de Ciência da computação, Centro de informática, Universidade federal de Pernambuco, Recife, 2017. Disponível em: <<https://www.cin.ufpe.br/~tg/2017-2/fmm4-tg.pdf>>. Acesso em: 18 nov. 2019.

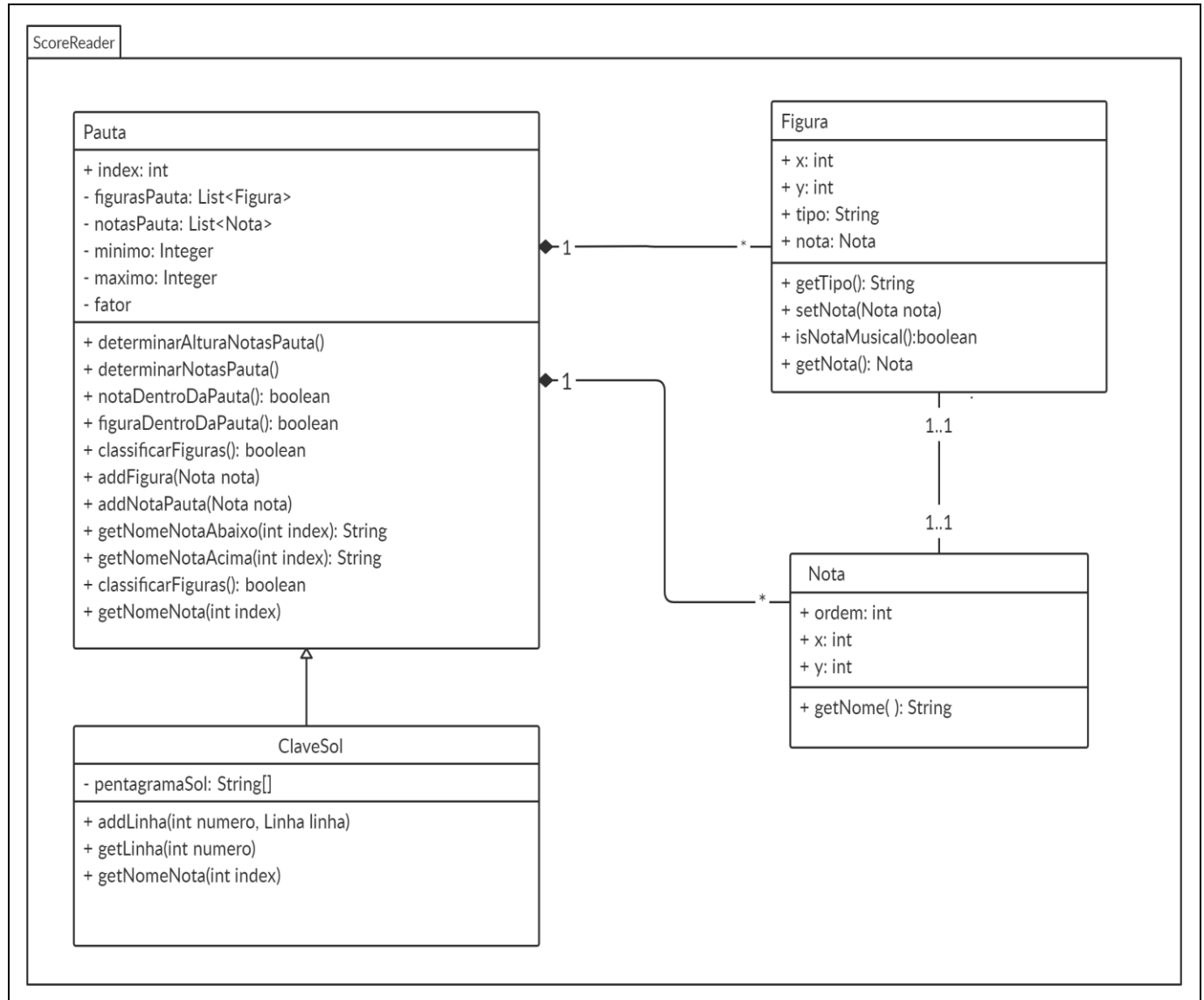


- MAURICENZ, Jonathan. **Interpres, um protótipo para reconhecimento de partitura:** Identificando a forma dos símbolos musicais. 2013. 57 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2013. Disponível em: <<http://campeche.inf.furb.br/tccs/BCC/2013-I/TCC2013-1-19-VF-JonathanMauricenz.pdf>>. Acesso em: 17 nov. 2019.
- MUSESCORE. **Musescore.** 01/12/2019. Disponível em: <<https://musescore.org/en>>. Acesso em: 01 dez. 2019.
- PACHECO, André. **K vizinhos mais próximos - KNN.** 2017. Disponível em: <<http://computacaointeligente.com.br/algoritmos/k-vizinhos-mais-proximos/>>. Acesso em: 29 nov. 2019.
- PEDREGOSA, Fabian et al. Scikit-learn: Machine Learning in Python. **Journal of machine learning research.** United States, p. 2825-2830. 12 out. 2011. Disponível em: <<http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>>. Acesso em: 17 nov. 2019
- QUEIROZ, José Eustáquio Rangel de; GOMES, Herman Martins. **Introdução ao processamento digital de imagens.** 2014. Disponível em: <<http://www.dsc.ufcg.edu.br/~hmg/disciplinas/graduacao/vc-2014.1/Rita-Tutorial-PDI.pdf>>. Acesso em: 18 nov. 2019.
- RUSSELL, Stuart; NORVIG, Peter. **Inteligência artificial.** 3. ed. Rio de Janeiro: Elsevier Editora, 2013. 998 p. Tradução de Regina Célia Simille de Macedo.
- SEUFERT, Andreas Dieter Mendes. **Investigação e aperfeiçoamento de algoritmos de reconhecimento de símbolos Musicais.** 2010. 47 f. Dissertação (Mestrado) - Curso de Engenharia Electrotécnica e de Computadores Major Telecomunicações, Faculdade de Engenharia da Universidade do Porto, Portugal, 2010. Disponível em: <<http://www.inescporto.pt/~jsc/students/2010AndreasSeufert/2010relatorioAndreasSeufert.pdf>>. Acesso em: 17 nov. 2019.
- SILVA, Gustavo Ântonny de Sousa da. **Reconhecimento automático de partituras manuscritas utilizando SVM e ORB.** 2017. 75 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luiz, 2017. Disponível em: <<https://monografias.ufma.br/jspui/bitstream/123456789/3563/1/GUSTAVO-SILVA.pdf>>. Acesso em: 29 nov. 2019.
- STANGE, Renata Luiza. **Adaptatividade em aprendizagem de máquina:** Conceitos e Estudo de Caso. 2011. 85 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Escola Politécnica da Universidade de São Paulo, São Paulo, 2011. Disponível em: <[https://www.teses.usp.br/teses/disponiveis/3/3141/tde-02072012-175054/publico/Dissertacao\\_RLStange\\_2011\\_Revisada.pdf](https://www.teses.usp.br/teses/disponiveis/3/3141/tde-02072012-175054/publico/Dissertacao_RLStange_2011_Revisada.pdf)>. Acesso em: 18 nov. 2019.
- WALSHAW, Chris. **Abc notation home page.** 1995. Disponível em: <<http://abcnotation.com/examples>>. Acesso em: 17 nov. 2019.

## APÊNDICE A – DIAGRAMA DE CLASSES CLIENTE

A Figura 25 mostra o diagrama classes contendo as principais classes e métodos existentes na parte cliente do protótipo desenvolvida linguagem de programação em Java.

Figura 25 – Diagrama das principais classes e métodos utilizados no cliente do protótipo

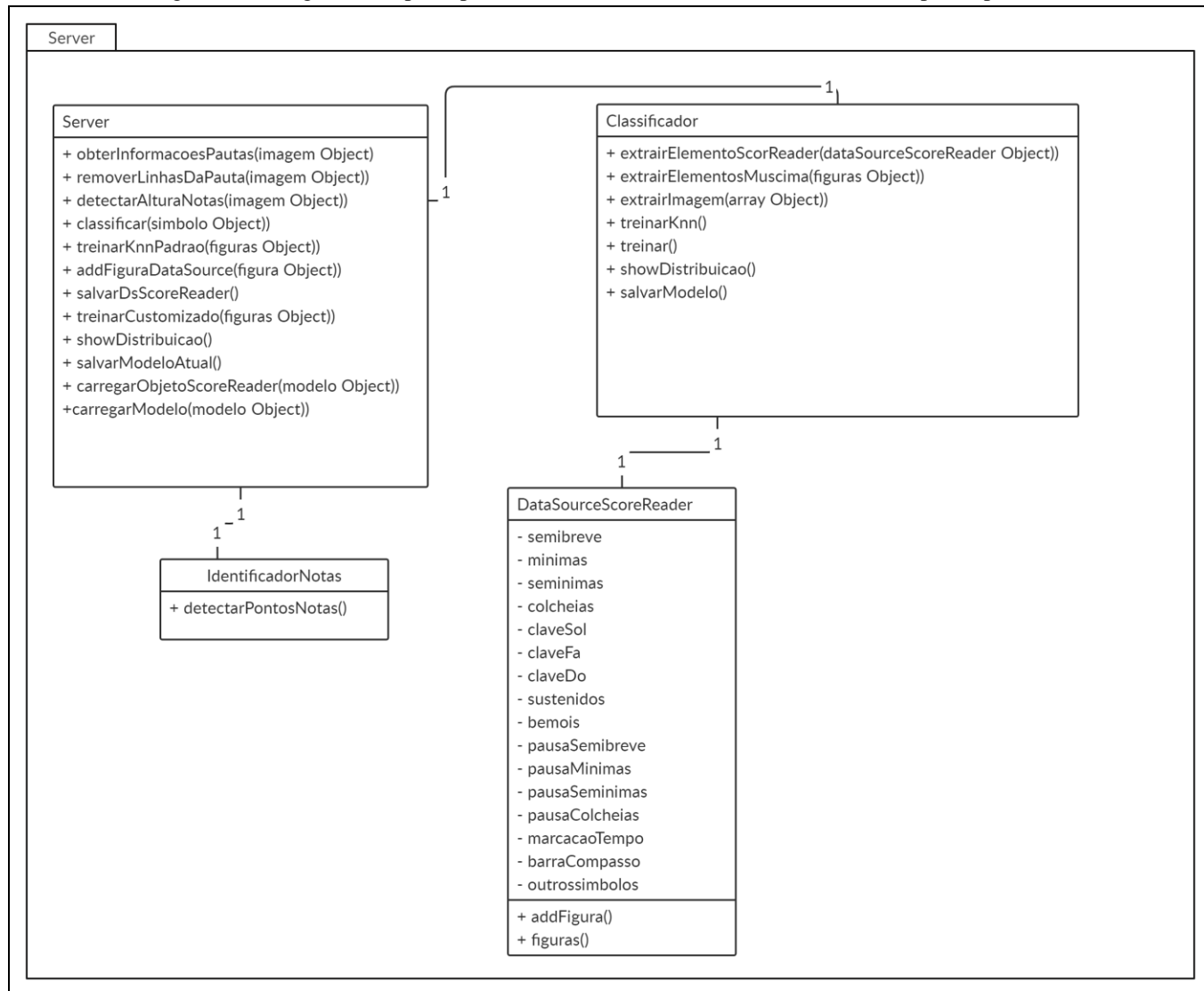


Fonte: elaborado pelo autor.

## APÊNDICE B – DIAGRAMA DE CLASSES SERVIDOR

A Figura 26 mostra o diagrama classes contendo as principais classes e métodos existentes na parte servidor do protótipo desenvolvida linguagem de programação em Python.

Figura 26 – Diagrama das principais classes e métodos utilizados no servidor do protótipo

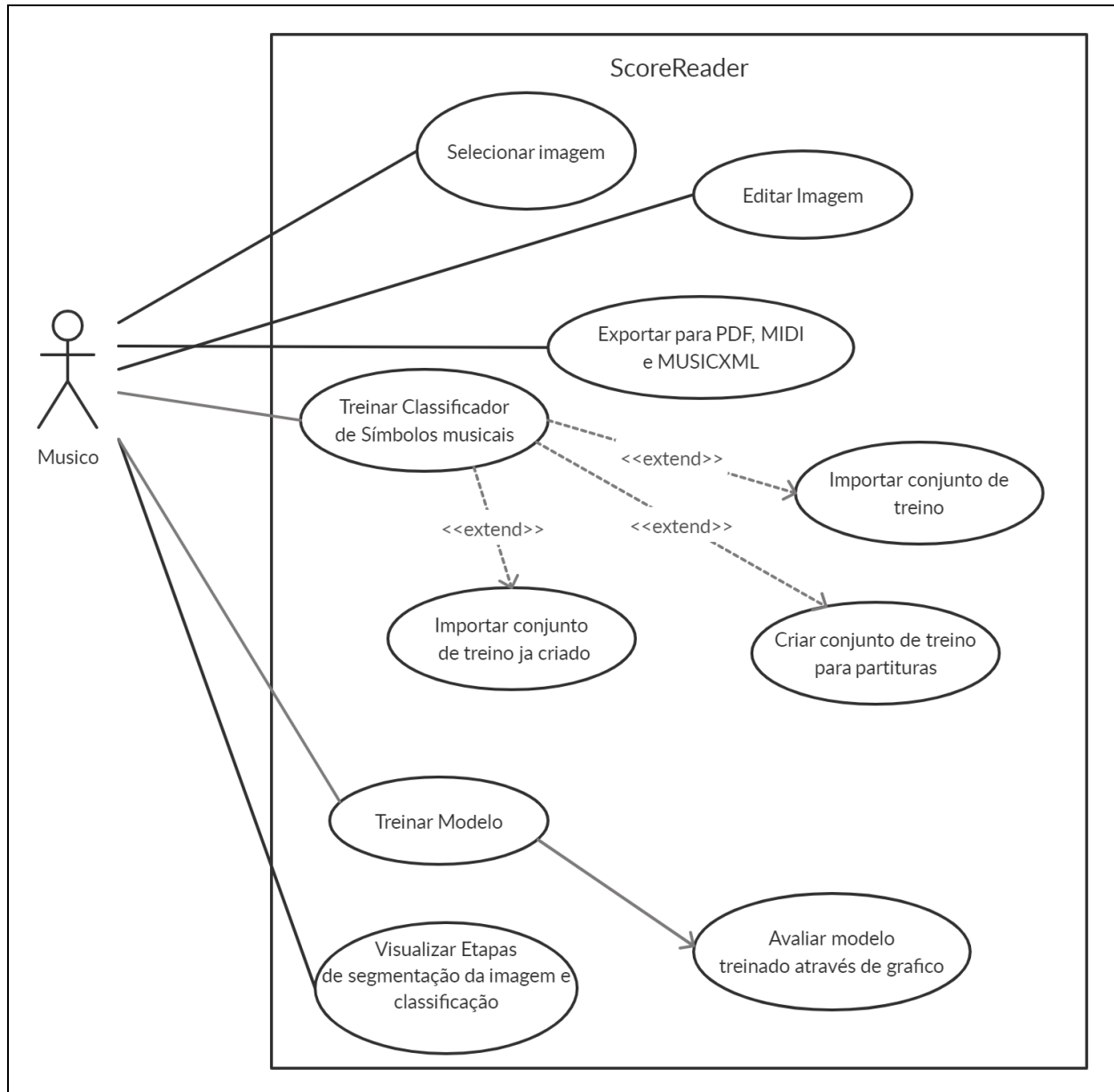


Fonte: elaborado pelo autor.

## APÊNDICE C – DIAGRAMA DE CASOS DE USO

A Figura 27 mostra o diagrama de caso de uso contendo as principais funcionalidades do protótipo.

Figura 27 – Caso de uso do protótipo




Fonte: elaborado pelo autor.

## APÊNDICE D – EXPERIMENTOS EM AMBOS PROTÓTIPOS

O Quadro 19 mostra a esquerda a imagem do experimento utilizado e a direita a comparação do resultado da saída do console em ambos protótipos.

Quadro 19 – Resultado de saída do console

Experimentos	Mauricenz (2013)	ScoreReader (2019)
	[Clave de sol, Seminima, Seminima, Seminima, Seminima]	Index: 0 Figura: ClaveSol Index: 1 Figura: Seminima Index: 2 Figura: Seminima Index: 3 Figura: Seminima Index: 4 Figura: Seminima
	[Clave de Do, Seminima, Pausa 1/1, Seminima, Colcheia, Pausa 1/2, Barra Compasso, Semibreve]	Index: 0 Figura: ClaveDo Index: 1 Figura: Seminima Index: 2 Figura: PausaSeminima Index: 3 Nota: g Figura: Seminima Index: 4 Figura: Colcheia Index: 5 Nota: B Figura: PausaColcheia Index: 6 Figura: BarraCompasso Index: 7 Nota: c Figura: Semibreve
	[Nao reconhecida, Nao reconhecida, Nao reconhecida]	Index: 0 Figura: MarcacaoTempo Index: 1 Figura: BarraCompasso Index: 2 Figura: MarcacaoTempo Index: 3 Figura: BarraCompasso Index: 4 Figura: MarcacaoTempo Index: 5 Figura: MarcacaoTempo
	[Clave de fa, Formula 44, Seminima, Pausa 1/1, Nao reconhecida, Pausa 1/2, Semicolcheia, Pausa 1/4, Colcheia, Barra Compasso, Nao reconhecida, Pausa 1/1, Colcheia, Pausa 1/2, Nao reconhecida, Pausa 1/4, Semicolcheia, Barra Compasso, Semibreve, Barra Compasso]	Index: 0 Figura: ClaveFa Index: 1 Figura: MarcacaoTempo Index: 2 Nota: e Figura: Seminima Index: 3 Figura: PausaSeminima Index: 4 Figura: Colcheia Index: 5 Nota: B Figura: PausaColcheia Index: 6 Figura: SemiColcheia Index: 7 Figura: PausaSemiColcheia Index: 8 Figura: Colcheia Index: 9 Figura: BarraCompasso Index: 10 Figura: Seminima Index: 11 Figura: PausaSeminima Index: 12 Figura: Colcheia Index: 13 Figura: PausaColcheia Index: 14 Figura: Colcheia Index: 15 Figura: PausaSemiColcheia Index: 16 Figura: SemiColcheia Index: 17 Figura: BarraCompasso Index: 18 Figura: Semibreve Index: 19 Figura: BarraCompasso

Fonte: elaborado pelo autor.

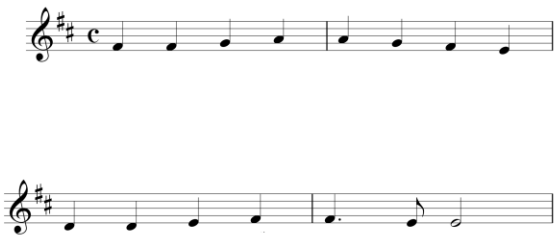
## APÊNDICE E – EXPERIMENTOS EM AMBOS PROTOTIPOS COM TRECHOS MUSICAIS

O Quadro 20 mostra a esquerda a imagem do experimento utilizado e a direita a comparação do resultado da saída do console em ambos protótipos.

### Quadro 20 – Resultado de saída do console

Música	Mauricenz (2013)	ScoreReader (2019)
	[Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Seminima,Nao reconhecida,Seminima,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Barra Compasso,Nao reconhecida,Seminima,Seminima,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Barra Compasso,Nao reconhecida,Nao reconhecida,Seminima,Seminima,Seminima,Nao reconhecida,Nao reconhecida,Nao reconhecida,Seminima,Seminima,Barra Compasso,Nao reconhecida,Nao reconhecida]	Index:0 Figura:ClaveSol Index:1 Figura:Sustenido Index:2 Figura:Sustenido Index:3 Nota:A Figura:Seminima Index:4 Nota:A Figura:Seminima Index:5 Nota:G Figura:Seminima Index:6 Nota:G Figura:Seminima Index:7 Figura:BarraCompasso Index:8 Nota:F Figura:Seminima Index:9 Nota:F Figura:Seminima Index:10 Nota:E Figura:Minima Index:11 Figura:BarraCompasso Index:12 Nota:D Figura:Seminima Index:13 Nota:D Figura:Seminima Index:14 Nota:A Figura:Seminima Index:15 Nota:A Figura:Seminima Index:16 Figura:BarraCompasso Index:17 Figura:ClaveSol Index:18 Figura:Sustenido Index:19 Figura:Sustenido Index:20 Nota:B Figura:Seminima Index:21 Nota:B Figura:Seminima Index:22 Figura:PausaSemiBreve Index:23 Figura:Minima Index:24 Figura:BarraCompasso Index:25 Nota:G Figura:Seminima Index:26 Nota:G Figura:Seminima Index:27 Nota:F Figura:Seminima Index:28 Nota:F Figura:Seminima Index:29 Figura:BarraCompasso Index:30 Nota:E Figura:Seminima Index:31 Nota:E Figura:Seminima Index:32 Nota:D Figura:Minima Index:33 Figura:BarraCompasso Index:34 Figura:BarraCompasso
	[Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao	Index:0 Figura:ClaveSol Index:1 Figura:Sustenido Index:2 Figura:Sustenido Index:3 Figura:MarcacaoTempo Index:4 Nota:F Figura:Seminima








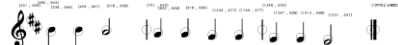





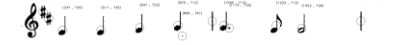
	reconhecida,Nao reconhecida,Semi nima,Nao reconhecida,Semi nima,Seminima,N ao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Nao reconhecida,Semi nima,Nao reconhecida,Semi nima,Seminima,N ao reconhecida,Nao reconhecida,Nao reconhecida]	Index:5 Nota:F Figura:Seminima Index:6 Nota:G Figura:Seminima Index:7 Nota:A Figura:Seminima Index:8 Figura:BarraCompasso Index:9 Nota:A Figura:Seminima Index:10 Nota:G Figura:Seminima Index:11 Nota:F Figura:Seminima Index:12 Nota:E Figura:Seminima Index:13 Figura:BarraCompasso Index:14 Figura:ClaveSol Index:15 Figura:Sustenido Index:16 Figura:Sustenido Index:17 Nota:D Figura:Seminima Index:18 Nota:D Figura:Seminima Index:19 Nota:E Figura:Seminima Index:20 Nota:F Figura:Seminima Index:21 Figura:BarraCompasso Index:22 Nota:F Figura:Seminima Index:23 Figura:BarraCompasso Index:24 Figura:Colcheia Index:25 Nota:E Figura:Minima Index:26 Figura:BarraCompasso
---	---	---

Fonte: elaborado pelo autor.

## APÊNDICE F – SEGMENTAÇÃO DOS SÍMBOLOS NO INTERPRES

O Quadro 21 mostra a esquerda a imagem utilizada para o experimento e a direita, o resultado da segmentação dos símbolos aplicados aos trechos musicais em ambos os protótipos. Observar-se que algumas hastes de símbolos na coluna “Interpres (2013)” foram removidas ou separadas em mais de uma parte, enquanto que na coluna “ScoreReader (2019)” os símbolos ficam mais nítidos. Uma vez que o símbolo foi deformado, o classificador não pôde determinar a o tipo do símbolo conforme explicado na Tabela 2.

Quadro 21 – Comparativo de segmentação de ambos protótipos aplicado a trechos musicais

Trecho Musical	Interpres (2013)	ScoreReader (2019)
 	 	 
 	 	 

Fonte: elaborado pelo autor.