Centro Universitário Católico Salesiano Auxilium - Araçatuba SP
UniSALESIANO

André Igor Gallacci
João Augusto Silva Lêdo
Leonardo César Bottaro

Composição de Partituras Automatizadas Através de *Machine Learning.*

Araçatuba
2017

André Igor Gallacci
João Augusto Silva Lêdo
Leonardo César Bottaro

Composição de Partituras Automatizadas Através de *Machine Learning.*

Relatório técnico apresentado como requisito parcial para obtenção de aprovação na disciplina Projeto de TCC, no Curso de Engenharia da Computação, no Centro Universitário Católico Salesiano Auxilium.

Araçatuba
2017

1. RESUMO

A partitura é uma linguagem musical padronizada, por um contexto histórico de representação universal. A partir desse contexto, a escrita de partituras utiliza-se da percepção musical e estudos específicos sobre a notação musical, ou seja sinais, símbolos e abreviações que compõem uma determinada partitura. Considerando que demanda um árduo processo de escrita manualmente; A fim de acelerar e automatizar o processo de criação de partituras, além de auxiliar a aprendizagem do manuseio de instrumentos musicais, desta forma é proposta uma ferramenta computacional que receba tais sonoridades musicais e através do emprego de inteligência artificial em *Machine Learning*, seja capaz de interpretar tais sonoridades musicais e fazer a escrita da partitura. Para tal realiza-se um levantamento bibliográfico focado em teoria musical, redes neurais e processamento digital de sinais e implementa-se em *JavaScript* e *Python* para o recebimento, processamento e apresentação do resultado. Os resultados obtidos são encorajadores, porém algumas limitações encontradas.

Palavras-Chave: Partitura, Música, Tempo Real, Inteligência Artificial, Aprendizagem de máquina.

2. ABSTRACT

The Music Sheet is a standard music language, with a universal representation backed by a historical context. From this context, the music notation utilizes the musical perception and specific research, meaning the use of symbols and abbreviations to compose a music score. Considering the complexity of writing such scores by hand, with the intent of accelerating, automating the process, and helping the learning stage, we propose a computer toolset that will receive, through a microphone, the sound of musical instruments, and then, by using machine learning, the computer should be able to correctly detect the music notes and then write them down as classic music notation. To this purpose, we got several references, including music theory, deep neural networks and digital signal processing, implemented in JavaScript and Python, to receive, process, and show the results. The results we obtained are encouraging, though some limitations were found.

Key-Words: Music Sheet, Music, Real Time, Artificial Intelligence, Machine Learning.

SUMÁRIO

3. INTRODUÇÃO

Atualmente no mercado, não quase não existem ferramentas de automação da composição de uma partitura que implemente *Machine Learning* para processar os dados recebidos das sonoridades em tempo real, ocasionando lentidão no processo de criação da partitura e em muitos casos imprecisão da composição de tal

Em razão da escassez no mercado de ferramentas de modo geral que utilizem *Machine Learning* (portanto um mercado praticamente inexplorado), surge a oportunidade de explorar os benefícios atuais que a inteligência artificial fornece na implementação de tais algoritmos que impulsionam a aplicação em questão adicionando a ela credibilidade, e uma maior exatidão, pelo fato de se tratar de um aprendizado de máquina no qual pode-se amenizar os possíveis erros existentes em uma aplicação convencional que não utilizam *Machine Learning* simplesmente aumentando a gama de aprendizado que a máquina irá receber.

A ideia principal do trabalho é suprir a escassez existente no mercado desse tipo de produto e propor uma ótica inovadora da maneira de se programar as futuras aplicações utilizando-se de métodos de *Machine Learning* e inteligência artificial, como boas práticas de futuras aplicações de qualidade no mercado.

O objetivo é alcançar com excelência e exatidão, ao final do cronograma da EAP proposta, um software interativo com uma interface gráfica simples e funcional que esteja conectada ao *TensorFlow*, o qual esteja recebendo adequadamente informações matriciais de um FFT devidamente programado para receber tais sonoridades musicais.

4. DESENVOLVIMENTO

A implementação do projeto demanda de um conjunto de ferramentas, para que se possa viabilizar e tornar-se possível sua concretização.

Inicialmente a captação do som, será feita pelos microfones do dispositivo que está a rodar a aplicação, o som deve ser captado e tratado através de um software desenvolvido em *JavaScript* o qual implementa a transformada de Fourier que irá transformar um sinal analógico em digital. Para fazer isso, a Transformada de Fourier analisa matematicamente a onda sonora e a transforma em espectros de um sinal discreto e, por conseguinte em amostras, para que o sinal analógico possa ser tratado digitalmente.

O software responsável pela digitalização do som denomina-se FFT (*Fast Fourier Transform*), ele deve ser implementado em *JavaScript* pois é a linguagem de programação que está sendo desenvolvido o software todo.O software, portanto, tem sua entrada definida, que é o som, e sua saída é uma matriz de 1024 amostras do som, a cada 1024 amostras, será gerado uma nova matriz, que inicialmente alimentará o *TensorFlow*, cujo qual é responsável pelo tratamento inteligente da aplicação.

O *TensorFlow*, é uma API da Google de código aberto que trabalha em seu interior com os métodos de *Deep-Learning*, atualmente propagado como um método extremamente promissor na área de Inteligência Artificial, a linguagem de programação e estruturação do *TensorFlow* para adequar seu sistema ao software em questão será o Python. O *TensorFlow*, em posse dessas matrizes trabalha em dois momentos distintos, o momento de seu treinamento, para que ele possa ter uma extrema eficácia na análise do som, e o momento em que ele estará de fato analisando as sonoridades, no primeiro momento será disponibilizado ao FFT sonoridades distintas, em seguida, será disponibilizado as notas musicais uma a uma em todas as suas oitavas, posteriormente esses dados serão retroalimentados incessantemente ao *TensorFlow* para que possa ser treinado adequadamente, o *TensorFlow* já devidamente configurado com os parâmetros de reconhecimento, em posse desses dados irá minimizar em cada ciclo de sua retroalimentação sua

margem de erro que é medido de 0 à 1, o treinamento só termina quando sua margem estiver em 0,99 e 1.

Finalizado a primeira etapa, a de treinamento, segue para a segunda etapa, a etapa principal, a de seu funcionamento, na qual o *TensorFlow*, recebe em tempo real as matrizes geradas pelo FFT cujo qual está a receber as sonoridades, e o *TensorFlow* interpreta essas matrizes, também em tempo real e a partir disso dará como saída números significativos que correspondem internamente no programa a um símbolo da partitura.

Posteriormente é chegada a fase visual do projeto, na qual irá disponibilizar a interface de interação entre usuário e o software, ela é desenvolvida em *JavaScript*. A parte visual irá receber os números significativos enviados pelo *TensorFlow*, e atribuir seus respectivos símbolos de partituras, criando um ambiente visual amistoso aos olhos de um músico ao enxergar tal partitura eletrônica cuja é executada em tempo real, recebendo os parâmetros do *TensorFlow*, convertendo-os notas em símbolos de partitura.

O motivo de se utilizar *JavaScript* como linguagem de programação oficial do projeto, é a possibilidade de se criar um projeto voltado aos navegadores web, dos quais nos permite uma portabilidade adequada quanto a multiplicidade de dispositivos, dos quais se pode rodar tais aplicações deixando uma brecha para futuras implementações como, a adequação do software como um modo de serviço online, alocado em um servidor. A utilização de Python se deve ao fato de o *TensorFlow* se utilizar desta linguagem para ser devidamente configurado, e a utilização do *TensorFlow* advém da disponibilidade de se utilizar uma ferramenta robusta que se utiliza de técnicas promissoras de inteligência artificial como o *Deep-Learning* para se implementar, fornecendo confiabilidade e proporcionando credibilidade satisfatória de um projeto digno de Engenharia, e de exímia qualidade para o mercado.

4.1. Ferramentas

4.1.1. Transformada de Fourier:

O método de cálculo da transformada discreta de Fourier a partir da expressão

Figura 1:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-i2\pi n \frac{k}{N}} = \sum_{K=0}^{N-1} f_k W^{kn} .n = 0.....N-1$$

Transformada de Fourier

Utiliza N^2 produtos entre números complexos e $N(N-1)$ somas, possuindo assim complexidade computacional $O\,(N\ log\ N)$ que está sendo implementado em *JavaScript*. O método utilizado é o FFT por dizimação em frequência cuja qual utiliza todas as dimensões do mesmo tamanho e calcula uma DFT de tamanho N.

4.1.2. Tempo da Música

O intervalo mínimo entre dois tempos de notas em milissegundos será calculado por:

$$IntervaloEmMs = \frac{m}{\frac{BPM*ms}{Tnota}}$$

É possível simplificar a fórmula em

$$IntervaloEmMs = \frac{m*ms}{Tnota*BPM}$$

Dado que:

m = 60 = quantidade de segundos em um minuto

ms = 1000 = quantidade de milissegundos em um segundo

Tnota = quantidade máxima desejada de notas tocadas em uma única batida

BPM = variável, de acordo com a música

Na aplicação, a quantidade máxima de notas em uma única batida é quatro (representada na partitura por uma semicolcheia)

Exemplo da fórmula acima aplicada aplicação:

$$IntervaloEmMs = \frac{60*1000}{4*BPM}$$

Ou

$$IntervaloEmMs = \frac{1500}{BPM}$$

Esse intervalo é utilizado para definir a frequência em que os dados serão enviados ao *TensorFlow* para o treinamento da rede neural.

4.1.3. Tensorflow

TensorFlow é uma ferramenta open source desenvolvida pelo Google em C++ com *endpoints* em Python, tem o intuito de impulsionar novas tecnologias e tendências no ramo da inteligência artificial para o consumo final, cujo qual tem a função de estimular e ajudar a criação de aplicações com uma aposta no futuro utilizando-se mais do que nunca aplicações práticas de inteligência artificial. Essa ferramenta está pautada em estudos de *Deep-Learning* e acredita-se ser o método mais promissor de se implementar uma inteligência artificial atualmente. Ele é responsável dentro do projeto por executar o back-end da Rede Neural, onde é feito todos os cálculos necessários de acordo com o modelo criado.

4.1.4. Python

O *Python* é uma linguagem de programação interpretada, nela existem diversas ferramentas matemáticas e um método inteligente de alocação de variáveis na memória o qual impulsiona e viabiliza implementações de inteligência artificial. A ferramenta *TensorFlow* está contida como uma biblioteca da linguagem *Python* para ser utilizada na criação dessas aplicações inteligentes.

4.1.5. JavaScript

O *JavaScript* é a linguagem eleita para se desenvolver toda a aplicação em questão, ela é uma linguagem com enfoque em aplicações web, e por conta disso está sendo implementada no projeto, por ela nos fornecer a desejada portabilidade, e permite executar a aplicação em qualquer sistema operacional e dispositivo que suportem um navegador moderno, assim portanto, aumentando a gama de consumidores, pois o *JavaScript* nos possibilita chegar a qualquer consumidor final que utilize qualquer sistema operacional em seus dispositivos.

4.1.6. Keras

É uma API de redes neurais artificiais de alto nível escrita em Python que é capaz de executar sobre o *TensorFlow* e tem a função de auxiliar o desenvolvimento no *TensorFlow* pois viabiliza uma melhor performance na construção do grafo da rede neural e diminui a quantidade de código boilerplate necessário.

4.1.7. VexFlow

É uma biblioteca Open-source criada em Javascript e disponibilizada gratuitamente no *gitHub* para a exibição de diversos tipos de notação musical, incluindo partituras convencionais

4.1.8. p5.js

É uma biblioteca *JavaScript* que tem o mesmo objetivo original do *Processing*: Fazer o código acessível a artistas, designers, professores e iniciantes, e reinterpreta isso para a *Web* de hoje. Utilizando a metáfora original de um “Software de desenho”, o p5 tem um *toolset* completo para desenho e renderização na web.

4.1.9. Tornado

É uma biblioteca para python com um *Web Framework* com funções de rede assíncronas, desenvolvida originalmente por *FriendFeed.* Utilizando Entradas e saídas não bloqueantes, a biblioteca pode ser escalada para dezenas de milhares

de conexões simultâneas, tornando-a ideal para respostas longas (*Long polling*) e conexões *WebSocket*.

4.1.10. WebSocket

O protocolo *WebSocket* permite a comunicação bidirecional entre um cliente executando código não-confiável em um ambiente controlado a um host remoto que optou por receber as comunicações de tal código. O protocolo consiste na abertura de um handshake seguido de um frame básico de mensagens, composto sobre uma camada TCP. O objetivo da tecnologia é providenciar um mecanismo para aplicações baseadas em navegadores que necessitam de comunicação bidirecional com servidores para que não dependam em abrir várias conexões HTTP para cada requisição feita. (MELNIKOV, 2011)

4.1.11. Player De Notas

Ferramenta desenvolvida em JavaScript que possui em sua programação a implementação do FFT (Fast Fourier Transform) cujo o qual em seu tempo de execução capta as sonoridades do microfone sincronicamente com a nota devidamente digitada para ser executada, ao fim da captação três arquivos são gerados dos quais serão utilizados no training set para configurar a rede neural, o FFT, a NotasTocadas e NotasNovas, o FFT como o próprio nome diz é um arquivo com n linhas contendo em contendo em cada linha 1024 amostras de som e suas respectivas energias e é utilizado posteriormente como entrada da rede neural, o arquivo de NotasTocadas é um arquivo com o mesmo número de linhas do FFT contendo as 1024 posições mas ao invés de serem preenchidas com as amostras de som, elas são preenchidas com zeros a ocorrência de um nesta linha ocorre quando é identificado a ocorrência de uma nota, esse arquivo é utilizado como saída desejada (Target da rede), já as NotasNova se tem a função de identificar se a nota está sendo sustentada ou se é uma nova nota (0 para notas sustentadas e 1 para notas novas).

4.1.12. MIDI.js

É uma biblioteca para *JavaScript* de código aberto, utilizado para a reprodução de arquivos tipo *MIDI* que são arquivos com sequências de notas e suas respectivas durações e alturas, comumente utilizado em instrumentos eletrônicos. A biblioteca facilita a criação de aplicativos *MIDI* na web, tanto players quanto aplicações interativas, inclui também uma biblioteca de instrumentos.

4.1.13. Softmax

É utilizado como função de ativação de uma camada da rede neural que possui como característica um mapeamento vetorial do plano em que se encontram os dados e faz um somatório vetorial entre as entradas e seus devidos pesos com o bias.

4.1.14. Relu

Função de ativação degrau cuja a qual faz o somatório clássico das entradas com o bias, com resposta linear a partir de 0.

4.2. Metodologia

4.2.1. Rede Neural (Aprendizado de máquina)

4.2.1.1. Player de notas

É o nome dado ao algoritmo que realiza a captação e a sincronização da entrada com a saída desejada da rede neural, para isso é utilizado a biblioteca MIDI.js para reproduzir os sons que são captados pelo algoritmo. a biblioteca permite tanto a execução de arquivos de musicas completo quanto a execução de notas arbitrárias, podendo assim gerar mais dados para alguma nota específica que esteja com um baixo nível de acurácia pela rede.

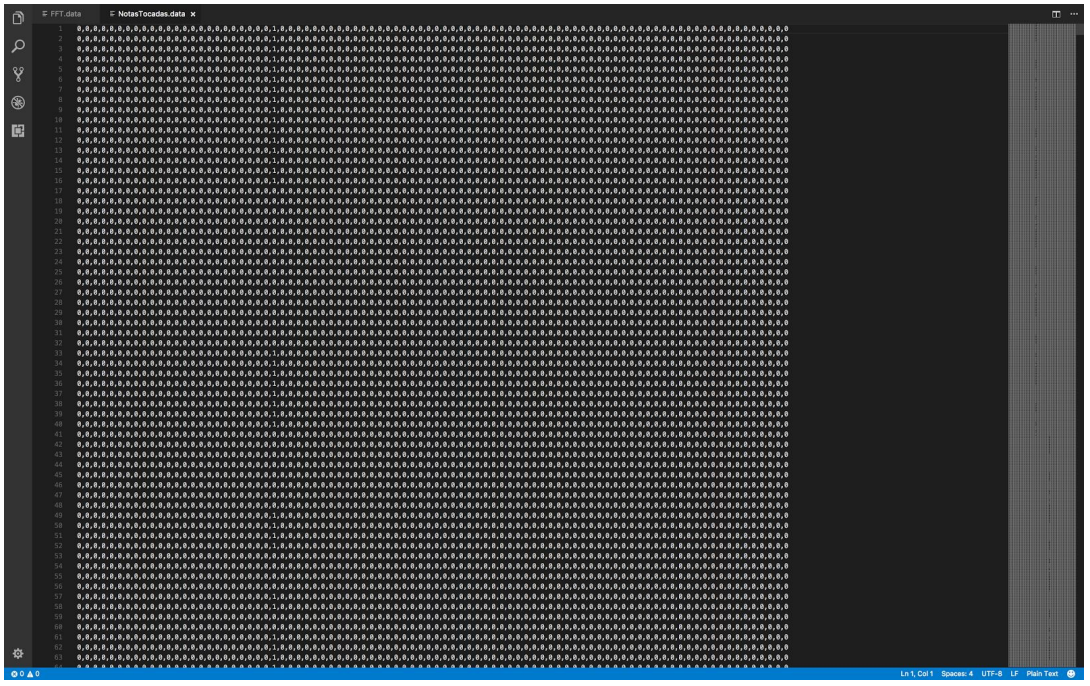
4.2.1.2. Captação dos dados para treinamento (*training set*):

É utilizado o Player de notas para gerar as amostras que são utilizadas no treinamento da rede neural, são digitadas as notas C3 (DÓ na 3ª oitava) até B5 (SI na 5ª oitava). Os arquivos gerados pelo player de notas são devidamente

concatenados para que cada linha do arquivo de FFT gerado pelo player corresponda a uma linha do arquivo de *NotasTocadas*, no ambiente de treinamento esses arquivos são evocados pela API Keras através da importação do *TensorFlow* como uma biblioteca. O Keras se utiliza do *TensorFlow* para montar o *Training Set* e indicar os arquivos de entrada e os arquivos de saída desejada.

Por fim, são gerados dois arquivos que contém dois vetores 88 posições representando os intervalos, o primeiro vetor de 88 posições contém as notas musicais que estão sendo tocadas no intervalo (0 para notas que não estão sendo tocadas e 1 para notas que estão sendo tocadas)

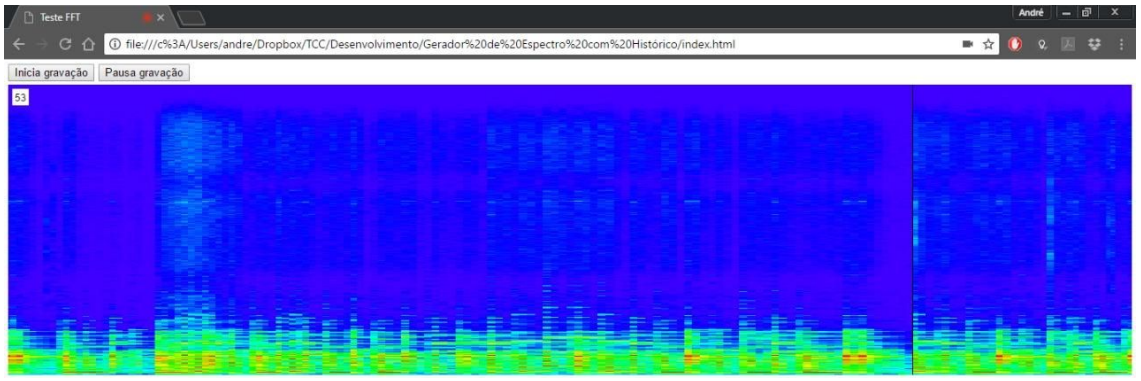
Figura 2:



Arquivo de saída desejada gerado a partir do MIDI.

Após a obtenção do arquivo com todos os intervalos do arquivo MIDI selecionado, foi reproduzido suas notas para serem captadas pelo microfone e processadas pelo FFT.

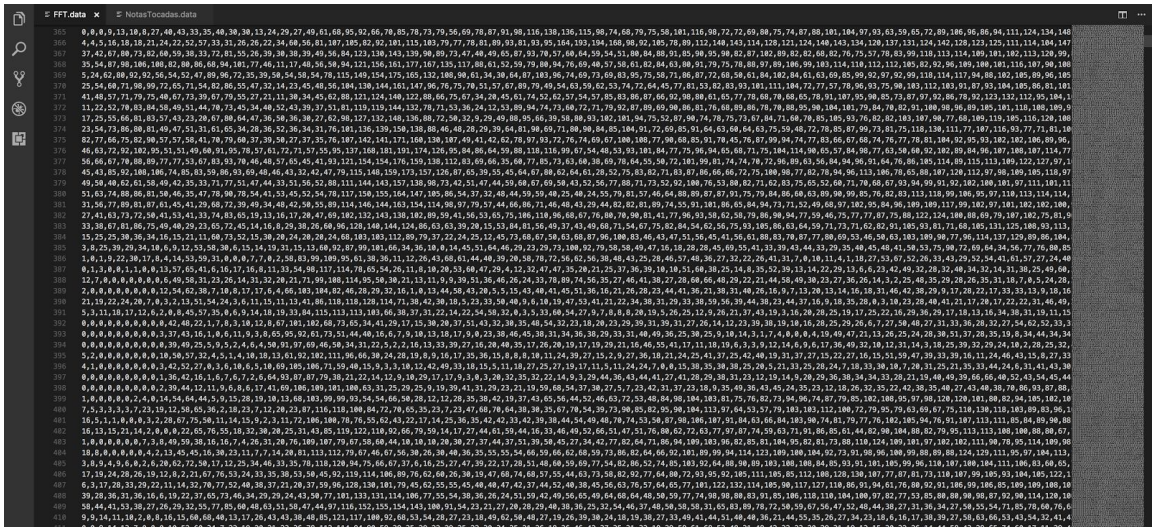
Figura 3:



Som do MIDI sendo analisado pelo FFT.

Após a análise, é gerado um arquivo (Figura 4) que contém o mesmo vetor de intervalos, porém, nesta etapa, cada intervalo é representado por um vetor de 1024 posições, até *6 KHz*, onde cada posição corresponde a quantidade de energia que há dentro uma faixa (0 é igual a nenhuma energia e 255 é igual a quantidade máxima de energia) o intervalo entre as frequências (resolução) é de $\frac{6000}{1024} = 5,86Hz$ aproximadamente.

Figura 4:



Arquivo gerado pelo FFT.

Todo este processo é repetido várias vezes para captar o mesmo som em diferentes ambientes e circunstâncias, depois todos os dados são concatenados em um único arquivo, gerando assim um *training set* rico em informações que pode ser

utilizado para o treinamento da rede. Para este projeto, foi gerado um *set* com aproximadamente 100.000 linhas de dados.

4.2.1.3. Modelo de rede neural utilizado:

Utilizamos 4 camadas, sendo que as camadas três primeiras estão configuradas com a função de ativação relu e a última camada com a função de ativação *softmax*. A primeira camada recebe a dimensão das linhas (1024) e as separa em pacotes de 512 para serem processados, a segunda camada reorganiza os pacotes recebidos da primeira camada em pacotes de 256, a terceira os capta da camada anterior com pacotes de dados de 176 amostras e por fim a quarta e última camada recebe os dados da camada anterior e os processa em pacotes de 88 amostras.

4.2.1.4. Treinamento da rede

Para o treinamento, O Keras foi configurado da seguinte maneira: Back-end de execução da Rede: *Tensorflow*. Para a função de otimização, utilizamos a "*adam*" pois ela utiliza de técnicas de "aceleração" no gradiente para chegar ao resultado correto mais rapidamente; a função de perda escolhida foi a "*binary crossentropy*" (cross-entropia binária) por conseguir tratar os erros de saídas binárias em um vetor de mais de uma dimensão. fizemos os treinamentos em "*batches*" (lotes) de 32 em 10 épocas, separando 10% dos dados de entrada para validação. Por fim, a métrica escolhida para monitoramento da performance da rede é a "*accuracy*" (precisão). A rede resultante é salva em formato ".h5" utilizando uma extensão do próprio Keras.

4.2.1.5. Execução da rede

Para executar a rede, ou seja, enviar um vetor de entrada que é o próprio som e obter a resposta em forma de probabilidade de notas, é executado um script em python que carrega a rede previamente treinada e salva em um arquivo, e então, utilizando-se do método "*predict"* do Keras, é passado o vetor de entrada, e a resposta "*callback*" da função é a saída da rede.

4.2.2. Servidor Python e Comunicação WebSocket

4.2.2.1. Servidor

É necessário um servidor onde as conexões, tanto da rede quando da interface gráfica, são centralizados. É utilizado então um simples servidor feito em Python com a biblioteca "*Tornado*". O código é extremamente simples: O servidor aguarda conexões, uma vez estabelecido, é feito um *request* para upgrade da conexão para o tipo *WebSocket*, após isso, tanto o cliente executando a interface quanto o servidor já estão aptos a enviar e receber dados. A comunicação é restrita ao usuário que fez a requisição, ou seja, se vários usuários simultâneos enviam dados, esses dados não se misturam.

4.2.2.2. Pacote do WebSocket

Com o intuito de manter o pacote leve, diminuindo assim sua latência de transmissão pela rede, os pacotes de requisição e resposta são minimalistas, sendo o conteúdo do pacote de requisição um simples vetor de 1024 dimensões separado por vírgulas (aproximadamente 1KB) e o conteúdo do pacote de resposta, dois vetores de 88 dimensões (172 Bytes)

4.2.2.3. Resposta do servidor

Para que o servidor consiga processar a entrada enviada pelo cliente, o método descrito acima em "Rede Neural - Execução da rede" é implementado dentro do script do servidor, no método *on_message* da biblioteca *Tornado.* Essa implementação faz com que o servidor execute o *Predict* do *Keras* para todos os pacotes recebidos do cliente, enviando como resposta a saída da rede.

4.2.3. Interface Gráfica (UI)

4.2.3.1. Javascript

Toda a interface gráfica é implementada utilizando *JavaScript* e bibliotecas auxiliares. Alguns métodos, como comunicação via *WebSocket*, utiliza-se de recursos do navegador e depende apenas da implementação do navegador, sendo

impossível o uso de bibliotecas para remediar a falta de tal recurso. Grande parte do projeto foi feito com o auxílio de duas bibliotecas principalmente, P5.js e *VexFlow*.

4.2.3.2. P5.js

Com o P5.js, é criado os algoritmos de captação de áudio do microfone e também o pré-processamento do Áudio que é enviado à rede, o FFT. Também é utilizado para a pré-visualização do áudio e da visualização das probabilidades resultantes da rede.

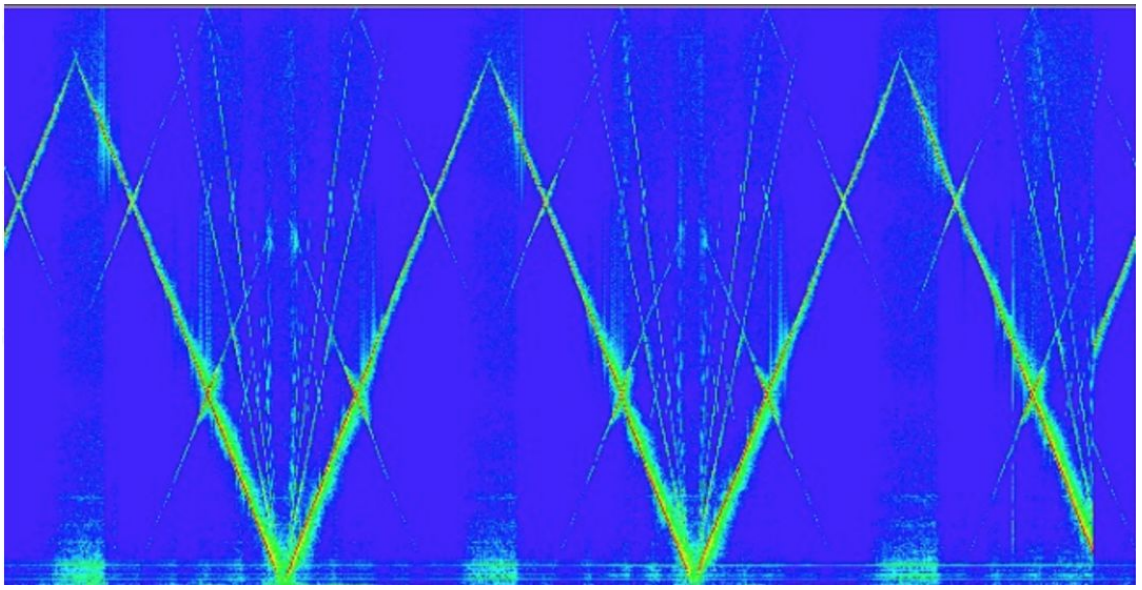
4.2.3.3. VexFlow

Utilizado para exibir os resultados da rede na forma de uma partitura musical clássica, os dados de resposta do servidor são processados e então dispostos à essa biblioteca de forma que a mesma consiga ler o resultado e escrever uma partitura coerente. Todo o processamento é feito em javascript puro e a biblioteca é utilizada somente no final de tudo para o desenho e exibição.

5. TESTES

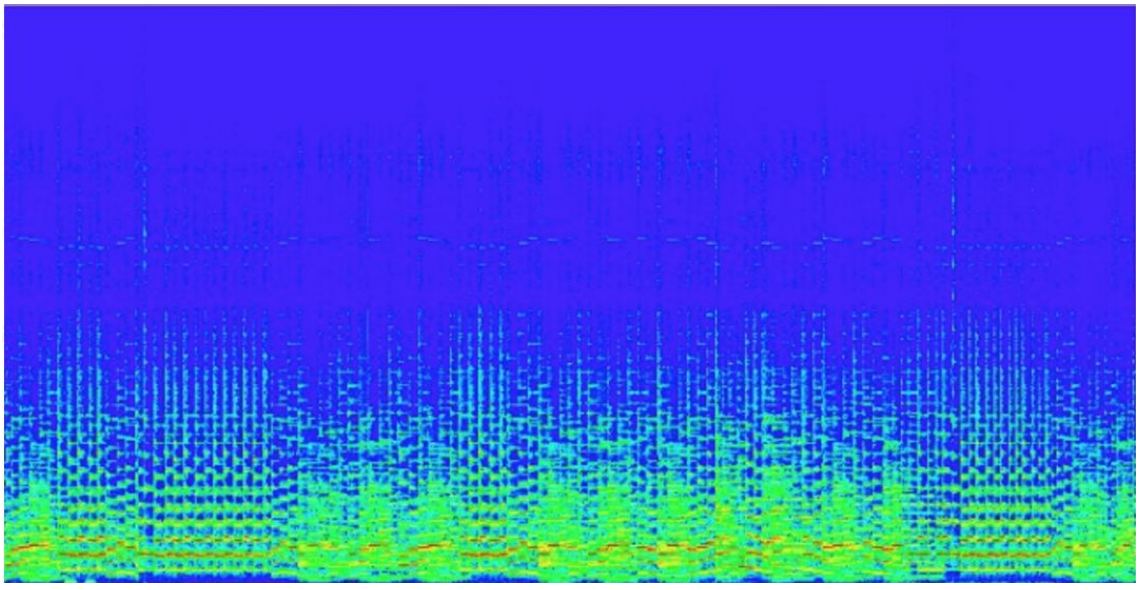
5.1. FFT (Fast Fourier Transform)

Figura 5:



Captação de 1Hz à 22kHz.

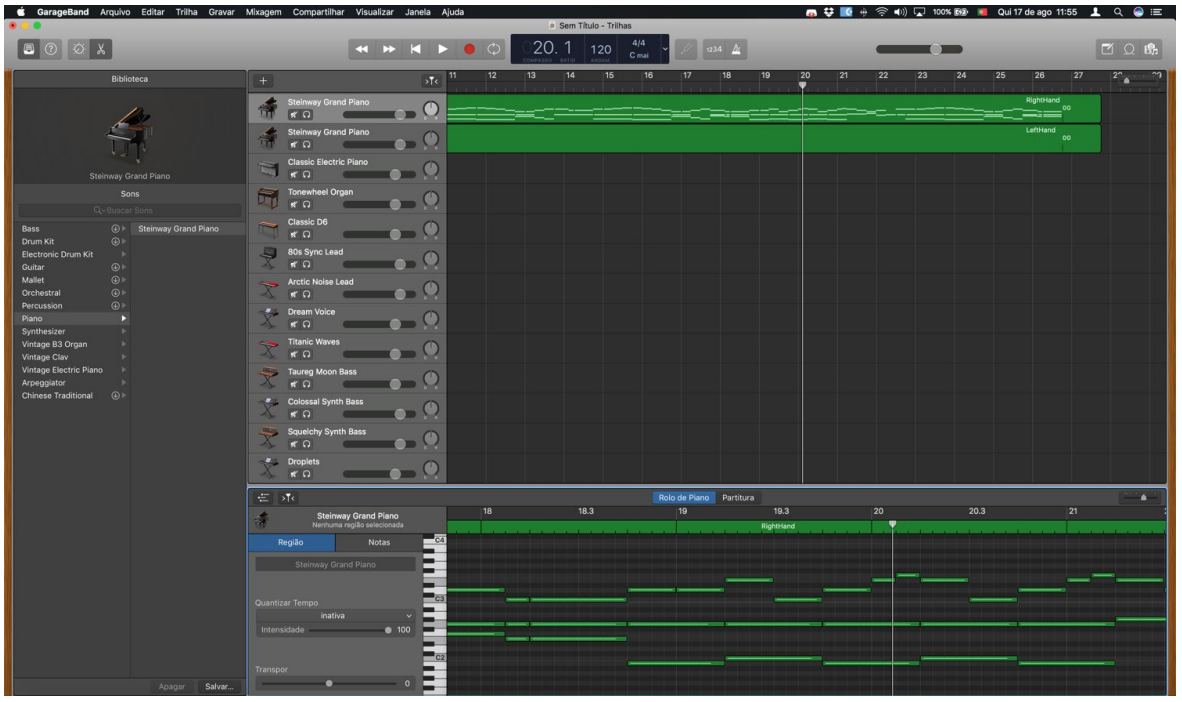
Figura 6:



Espectro de uma música

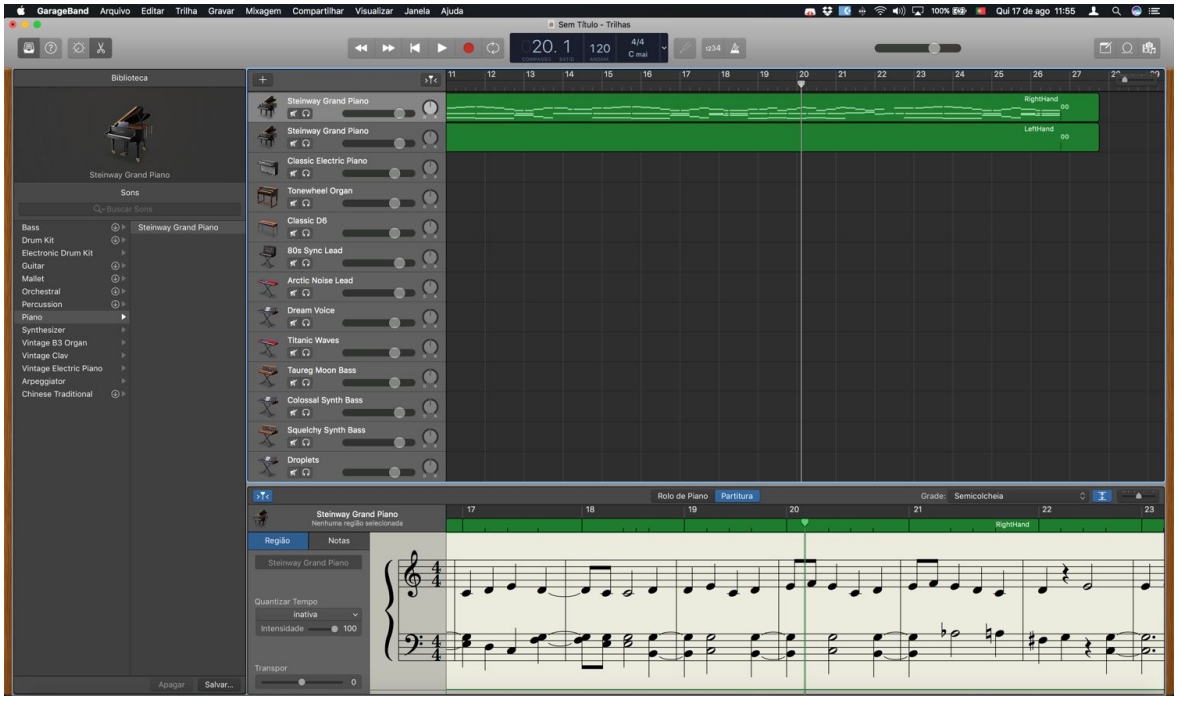
5.2. MIDI e Resultado esperado

Figura 11:



Leitura do MIDI e espaçamentos.

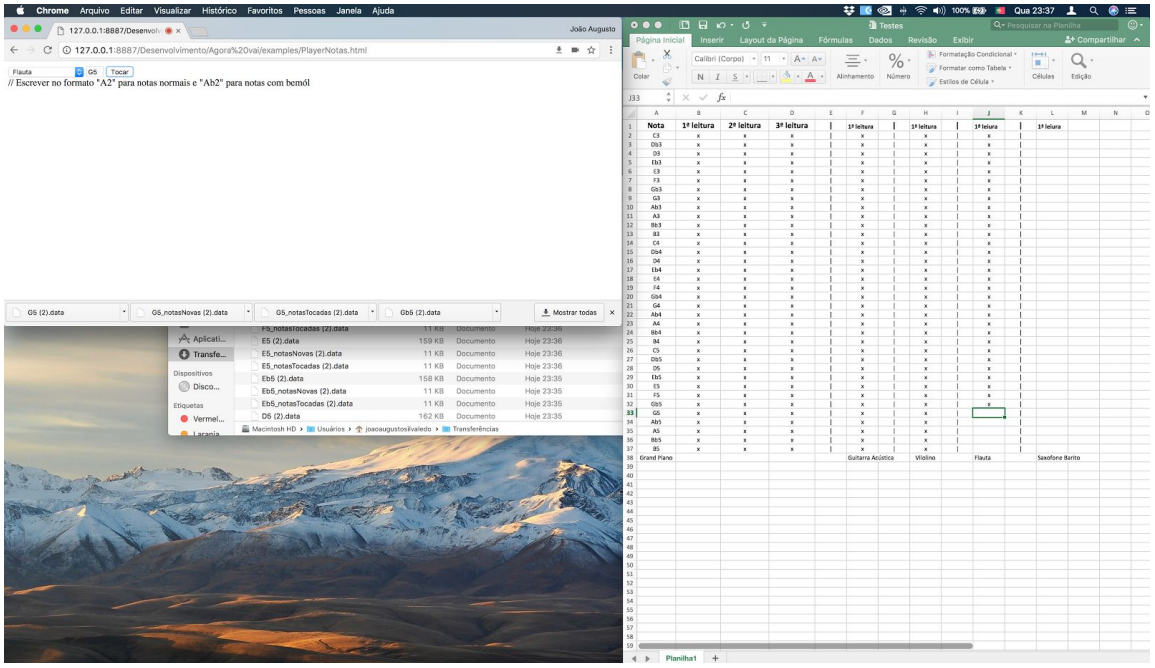
Figura 12:



Leitura do MIDI convertido em sua partitura correta (saída desejada).

5.3. Captação dos dados

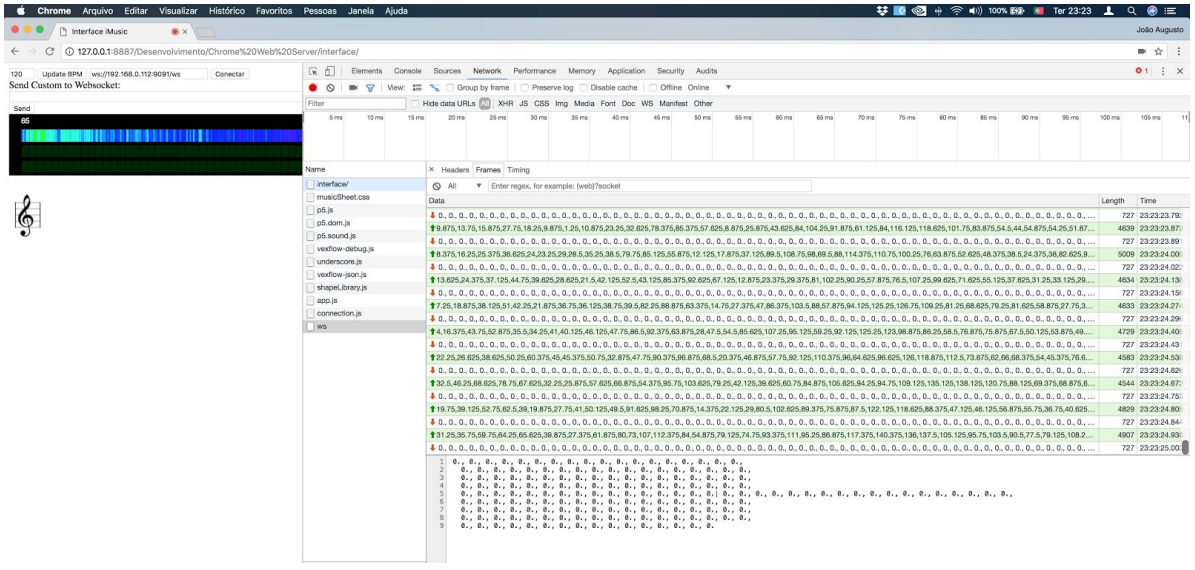
Figura 7:



Execução do Player de Notas para captação de dados para o treinamento da rede neural.

5.4. Comunicação WebSocket

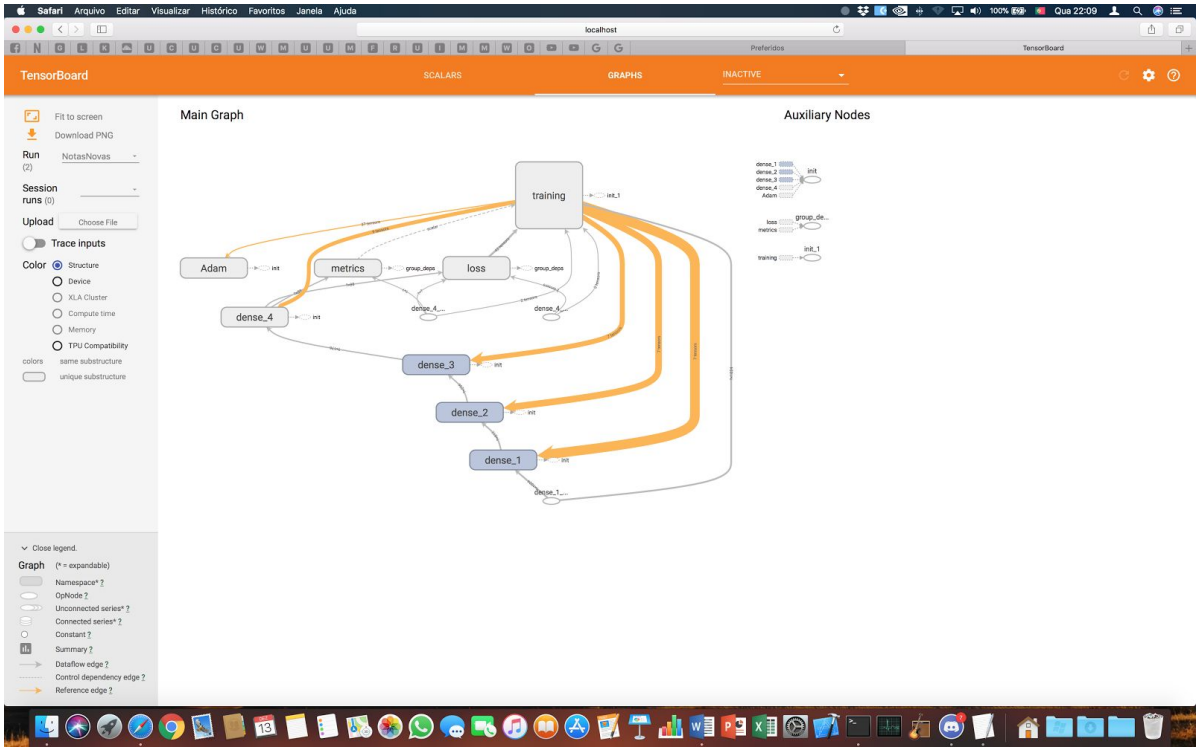
Figura 8:



Conteúdo da comunicação entre a interface gráfica e o servidor da rede neural através de pacotes *WebSocket*.

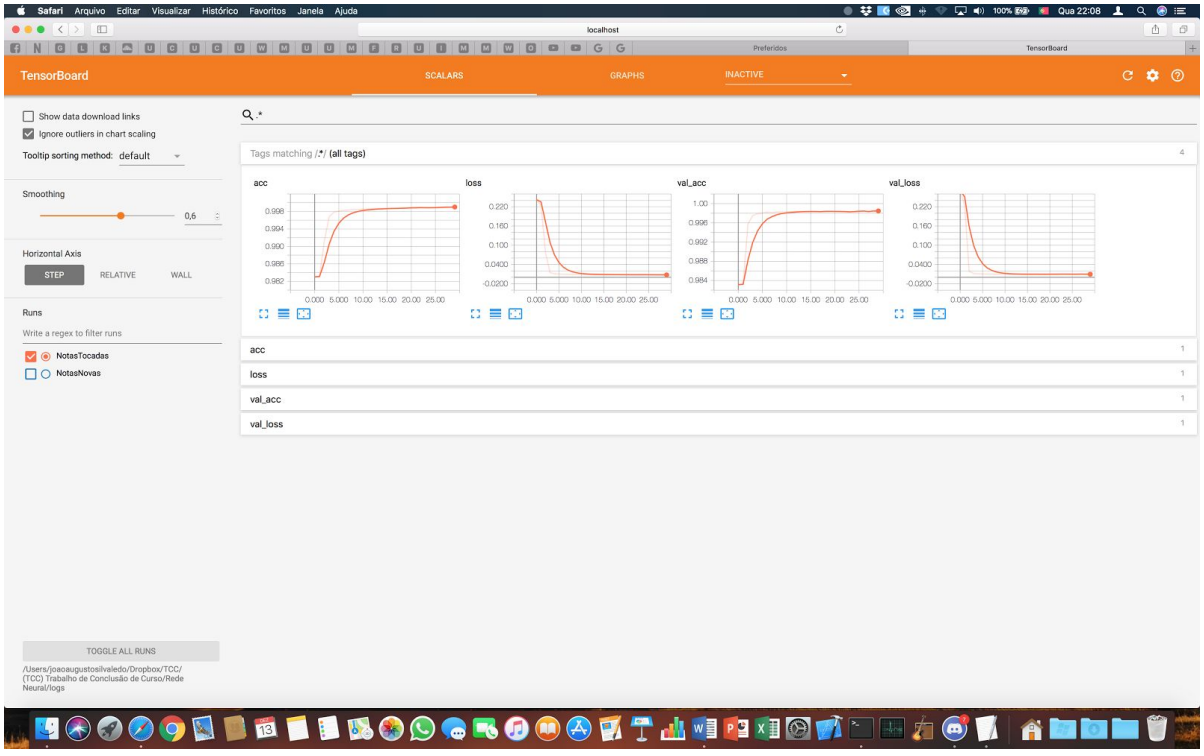
5.5. Rede Neural

Figura 9:



Grafo da Rede Neural.

Figura 10:



Gráficos do comportamento da Rede Neural realizado pelo TensorBoard.

6. CONCLUSÃO

O objetivo apresentado foi parcialmente atingido. A Rede Neural produzida é capaz de identificar notas individuais e suas respectivas pausas. O ambiente gráfico é capaz exibir a representação do que foi captado e identificado pela rede.

A principal limitação do projeto é a inabilidade de reconhecer múltiplas notas simultaneamente. Por conta da dificuldade presente na baixa resolução do FFT, a rede neural não foi capaz de abstrair corretamente o que compõe uma nota, sendo assim, a rede resultante desta pesquisa consegue somente reconhecer notas singulares.

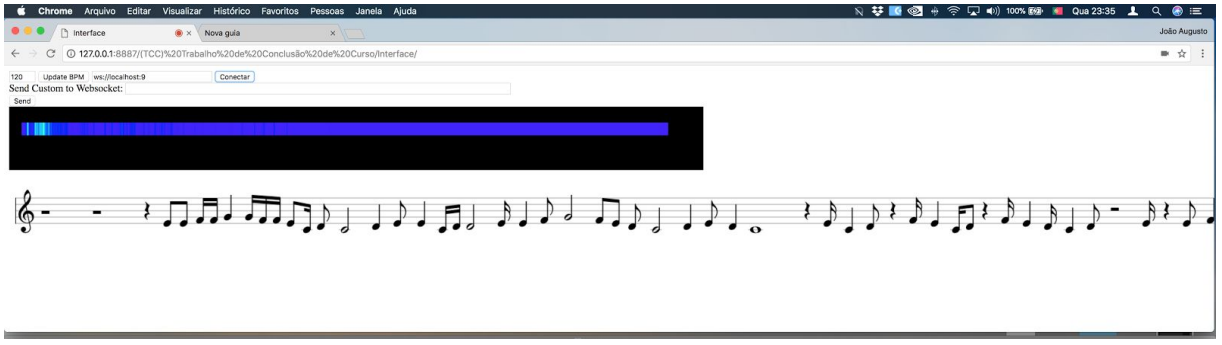
O uso do FFT antes da Rede Neural pode ser o principal responsável por essa inabilidade. O uso das sonoridades diretamente na rede, sem um pré-processamento, junto de um método espacial para reconhecimento dos sons, provavelmente otimizaria o reconhecimento, pois o fator limitador é o próprio FFT.

A coleta de mais dados para o treinamento também é essencial para o aperfeiçoamento para a Rede Neural. Juntamente com mais variedades de instrumentos, ambientes, microfones, e caixas de som, providenciando assim maior qualidade e variedade de dados que representam o mesmo conceito.

Dado que o foco desta pesquisa é a composição de partituras automatizadas e não somente de *Machine Learning*, mais recursos deverão ser investidos na pesquisa da Rede Neural detectora de notas.

Todo o código fonte e recursos utilizados no projeto estão disponíveis em: https://github.com/leocb/Composicao-Automatica-de-Partituras-com-Machine-Learning.

Figura 11:



Resultado final obtido com a rede neural, exibido na interface gráfica do projeto.

7. REFERÊNCIAS

CATALOGAÇÃO DE MÚSICA IMPRESSA. <http://www.abinia.org/catalogadores/56-181-1-PB.pdf > Acessado em 4 de outubro de 2017.

DEEP LEARNING ...MOVING BEYOND SHALLOW MACHINE LEARNING SINCE 2006. <http://deeplearning.net> Acessado em 4 de outubro de 2017.

ICMC-USP. APRENDIZADO DE MÁQUINA. <http://www2.ic.uff.br/~kdmile/MachineLearning_Andre.pdf> Acessado em 4 de outubro de 2017.

IME-USP. TRANSFORMADA DE FOURIER: FUNDAMENTOS MATEMÁTICOS, IMPLEMENTAÇÃO E APLICAÇÕES MUSICAIS. <https://www.ime.usp.br/~kon/MAC5900/seminarios/seminario_Jorge.pdf>

Acessado em 11 de maio de 2017.

JAVASCRIPT. *START CREATING THE FUTURE NOW.* <https://www.javascript.com> Acessado em 29 de abril de 2017.

MIDI.js. <https://galactic.ink/midi-js/> Acessado em 15 de agisto de 2017.

P5.js. <https://p5js.org/> Acessado em 05 de abril de 2017.

PYTHON. *PYTHON IS A PROGRAMMING LANGUAGE THAT LETS YOU WORK QUICKLY AND INTEGRATE SYSTEMS MORE EFFECTIVELY.* <https://www.python.org> Acessado em 3 de maio de 2017.

TENSORFLOW. *AN OPEN-SOURCE SOFTWARE LIRARY FOR MACHINE INTELIGENCE.* <https://www.tensorflow.org> Acessado em 3 de maio de 2017.

TORNADO. <http://www.tornadoweb.org> Acessado em 19 de julho de 2017.

KERAS: THE PYTHON DEEP LEARNING LIBRARY. <https://keras.io> Acessado em 29 de setembro de 2017.

THE WEBSOCKET PROTOCOL < https://tools.ietf.org/html/rfc6455 > Acessado em 27 de junho de 2017.

WEBSOCKET. *THIS IS WEBSOCKET.ORG.* <https://www.websocket.org> Acessado em 29 de abril de 2017.