UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final

Trabalho de Conclusão de Curso

Plataforma móvel para coleta de dados e treinamento de redes neurais em nuvem

Autor: **Bruno Eduardo Santos de Oliveira**

Orientador: **Prof. Dr. Josué Labaki Silva**

Campinas, Janeiro de 2021

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final

Trabalho de Conclusão de Curso

Plataforma móvel para coleta de dados e treinamento de redes neurais em nuvem

Autor: **Bruno Eduardo Santos de Oliveira**
Orientador: **Prof. Dr. Josué Labaki Silva**

Curso: Engenharia de Controle e Automação

Trabalho de Conclusão de Curso apresentado à Comissão de Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Engenheiro de Controle e Automação.

Campinas, 2021

S.P. - Brasil

Dedicatória

Dedico este trabalho aos meus pais, Carlos e Matilde, por todo suporte durante minha graduação.

Agradecimentos

Este trabalho não poderia ser concluído sem a ajuda de diversas pessoas que contribuíram direta ou indiretamente, às quais presto essa homenagem.

Agradeço ao orientador deste trabalho, Prof. Dr. Josué Labaki Silva, pelo auxílio no entendimento da teoria da propagação de ondas mecânicas e como estas poderiam ser analisadas por aprendizado de máquina.

Agradeço a comunidade do *Tensorflow*, que compartilha diversos conhecimentos sobre *Deep Learning* em fóruns. Esta união e vontade de desenvolver soluções é a quem atribuo o sucesso e popularização do conhecimento desta área de conhecimento.

Agradeço a todos os professores que direta ou indiretamente contribuíram para o desenvolvimento deste trabalho, compartilhando conhecimentos nas áreas correlacionadas.

Agradeço ao CNPq e a Unicamp que, juntamente, contribuíram com o financiamento da Iniciação Científica que precede este trabalho pelo projeto PIBIC.

Sumário

Resumo

Este Trabalho de Graduação tem por objetivo desenvolver uma plataforma móvel para coleta de dados e treinamento de redes neurais em nuvem. Como exemplo de aplicação, considera-se inicialmente um modelo de *Deep Learning* para identificação das teclas pressionadas por um teclado, através da vibração captada por um *smartphone*, posicionado próximo ao teclado.

Portanto, parte do Trabalho foi desenvolver um modelo de identificação de teclas para verificar a aplicação da plataforma utilizando, a princípio, dados coletados em um acelerômetro e sistema de aquisição profissionais.

Em seguida foi desenvolvido um aplicativo para a coleta da vibração captada pelo acelerômetro de um *smartphone* e envio destes dados a um servidor *cloud* cuja configuração também é parte deste Trabalho.

O servidor recebe os dados do *smartphone* e envia a classificação da tecla para o dispositivo. A configuração foi feita de forma modular para acomodar modificações pertinentes a outros projetos que se beneficiem da vibração captada pelo *smartphone*, mas não sejam classificadores de teclas.

Lista de Figuras

1 Introdução

Com a popularização da computação pessoal e o barateamento do custo de processamento de dados, algoritmos de treinamento de máquina se tornaram cada vez mais populares e, consequentemente, mais eficientes conforme o aumento de pesquisas por otimizações e novas técnicas de aprendizado de máquina.

Dentre as novas técnicas, destacou-se popularmente o *Deep Learning* que consegue boa acurácia em troca de mais processamento e mais dados de treinamento para que os padrões sejam reconhecidos (Banko, 2001).

Uma das característica do *Deep Learning* é conseguir fazer inferências abstratas em dados muito brutos. Por exemplo, a identificação do que está presente numa fotografia e até mesmo identificar quem está presente nesta fotografia. A rede é capaz de fazer inferências tendo apenas como entrada os valores RGB de cada pixel de uma foto ou equivalente em outros padrões de armazenamento de imagens.

Em alguns casos não é conveniente utilizar uma única plataforma para treinar uma rede de aprendizado de máquina, desde a extração de dados até a inferência, visto que o custo computacional é grande. Por exemplo, um *smartphone* possui muita informação de seu usuário, mas possui pouco processamento e memória alocável, para uma única aplicação treinar uma rede com dezenas de milhares de nós. Em contra-partida, um servidor dedicado pode ser alugado com a finalidade única de receber os dados dos *smartphones* e fazer exclusivamente as etapas de processamento, utilizando, por exemplo, GPUs e TPUs.

A popularização dos *smartphones* também aumentou na última década, aumentando a quantidade de pessoas possivelmente expostas a um ataque por aplicações maliciosas que se instalem nos dispositivos. Entretanto, diversas medidas de segurança são criadas e enviadas aos usuário finais como forma de garantir, por exemplo, a segurança de dados sensíveis.

Grande parte das investigações de seguranças são feitas a nível de *software*,

de forma que uma nova categoria de vulnerabilidades, chamada de *Side-channel* foi descoberta. Esta nova categoria de vulnerabilidade abusa de acessórios paralelos as informações sensíveis, como o tempo de processamento foi utilizado por Lipp et. al. (2018) para ler dados em espaço de memória restrito ao *Kernel* por aplicações a nível de usuário. Utilizar recursos de *Hardware* para extrair informações indevidas também é considerado um ataque *Side-channel*.

Diversas medidas de seguranças já são aplicadas nos *smartphones* para acesso a recursos de *hardware* por aplicações terceiras. Um exemplo comum é a mensagem de permissão de acesso a recursos em celulares Android, conforme a figura 1, onde uma janela de permissão é apresentada ao usuário ao abrir um aplicativo que precise acessar a câmera pela primeira vez.

Figura 1: Janela de permissão de acesso a um recurso no sistema operacional Android

Entretanto, alguns recursos podem ser utilizados pelos aplicativos sem que uma permissão de acesso seja requisitada ao usuário. Alguns destes recursos são essenciais ao aplicativo, como utilização do processador e memória em espaço de usuário. Outros destes recursos que não precisam de autorização, são necessários apenas em atividades específicas, como o GPS e giroscópio.

Nos sistemas Android e iOS o acesso aos sensores pode ser feito por qualquer aplicação instalada, sem a necessidade de autorização do usuário e para qualquer taxa de amostragem do sensor, conforme sua documentação (Acesso em Dezembro de 2020). Owusu et. al. (2012) já demonstraram que é possível obter posições de

toque na tela apenas pelo acesso ao acelerômetro. Desta forma uma aplicação em plano de fundo pode inferir os toques a tela realizados em uma aplicação no plano principal.

Berger (2006) e Marquardt (2011) mostraram que pode-se inferir quais teclas foram pressionadas em um teclado próximo a um *smartphone* através do microfone ou acelerômetro do dispositivo. Entretanto não propuseram uma abordagem onde a identificação de todos os caracteres fosse o objetivo da inferência. Berger obteve um bom resultado para combinações de caracteres em que houvesse um grande número de repetição, enquanto Marquardt se utilizou de um dicionario de palavras, de origem inglesa, para auxílio do treinamento da rede.

Entretanto, senhas não precisam estar atreladas a palavras pertencentes a um dicionário, possuírem padrão de repetição e podem ser uma combinação aleatória de caracteres e a pesquisa prévia a este trabalho foi focada na identificação de senhas (utilizando a vibração captada por um acelerômetro profissional) e será discutida na revisão bibliográfica.

Ainda assim, não é possível inferir que esta pesquisa prévia é capaz de classificar dados captados por um *smartphone*, visto que os dados foram captados por um acelerômetro profissional. Este trabalho desenvolve uma plataforma para captação da vibração mecânica do acelerômetro direto de um *smartphone* para uma rede de *Deep Learning* já treinada para que esta faça a identificação das teclas pressionadas em um teclado próximo e retorne ao dispositivo qual tecla foi pressionada.

2 Revisão Bibliográfica

Previamente a este Trabalho de Graduação foi realizado uma Iniciação Científica para inferir caracteres pressionados em um teclado utilizando os sinais de vibração captado por um acelerômetro com sistema de aquisição profissional. Os tópicos das pesquisas realizadas que são pertinentes a criação da plataforma de coleta de dados são apresentados, juntamente com as justificativas de sua relevância à plataforma.

Inicialmente foi feito um aprofundamento sobre *Machine Learning* e *Deep Learning*, visto que um entendimento da primeira contribui com conceitos que são base da segunda. Em seguida foi feito uma revisão sobre o estado da arte de *Keylogging* utilizando *smartphones* e a arquitetura de acesso aos sensores dos dispositivos. Em seguida foi realizado uma pesquisa sobre arquiteturas em *cloud*, que serão utilizados na metodologia deste Trabalho.

2.1 Deep Learning

Métodos de Deep Learning são métodos de aprendizagem por representação de muitas camadas de abstração simples (LeCun, Y., Y. Bengio, and G. Hinton, 2015). Métodos de representação são formas de aprendizado de máquina onde não existe muito pré-processamento do dado de entrada, ou seja, um dado bruto é inserido em um algoritmo e este é capaz de classificá-lo.

A classificação é feita a partir de um conjunto de amostras, que são separados em dois sub-conjuntos: conjunto de treinamento e conjunto de avaliação. Cada amostra é composta por um grupo de características (ou *features*) e um único valor, ou rótulo, correspondente. Por exemplo, uma imagem de um cachorro de resolução de 256x256 possui 65536 pixeis, ou seja, 65536 *features* e seu rotulo é "Cachorro".

Um conjunto de treinamento tem por objetivo obter a melhor função que mapeie

cada grupo de *features*, de cada amostra, para seu rótulo correspondente. Isso é feito através de uma estrutura de camadas, um *solver* e uma função de custo. O *solver*, ou otimizador, é um algoritmo iterativo que tende a diminuir um valor, denominado custo ou *loss*. O *solver* "Método do gradiente"é apresentado em 1 como exemplo, e minimiza o custo $J(\theta_0, \theta_1)$, definida em 2.

$$\textit{Repita até a convergência:}\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{1}$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2}$$

θ é o vetor de pesos de toda a rede neural.

α é a taxa de aprendizado da rede

m é a quantidade de amostras no grupo de treinamento

$\hat{y}$ é a saída estimada pela rede

y é a saída esperada

O sub-conjunto de avaliação tem por objetivo verificar qual o custo (neste caso é usualmente chamado de erro) em amostras que não foram usadas no sub-conjunto de treinamento. Isto mostra como o modelo se comportaria para um dado novo.

A quantidade de amostras do conjunto de treinamento é fator fundamental para a acurácia de um algoritmo de aprendizado de máquina (Banko, 2001). Isto é ainda mais relevante quando abordamos Deep Learning, visto que esta possui ainda mais camadas de abstração que pode sofrer *overfitting*. Sendo assim, o conjunto de amostras utilizado numa abordagem com Deep learning é usualmente maior que a quantidade utilizada em um aprendizado de máquina tradicional.

Como existe uma separação do conjunto de amostras em dois sub-conjuntos,

observa-se que resultados diferentes podem ser obtidos para cada separação, dado que a separação dos elementos é pseudo-randômica. Uma solução para essa inconsistência de resultados é a técnica K-fold que reparte o conjunto de amostras em K sub-conjuntos. São realizados K execuções do treinamento, utilizando apenas um sub-conjunto para avaliação e uma única vez. Assim todos os elementos estarão no conjunto de avaliação apenas uma vez entre as K execuções. Com isso, obtemos K valores de acurácia e pode-se fazer uma analise estatística em relação ao conjunto de acurácias, diminuindo o enviesamento da separação dos sub-conjuntos.

2.2 Tensorflow e Keras

Na iniciação científica que precede este Trabalho da Graduação, algumas plataformas de desenvolvimento para Deep Learning foram consideras. O pacote Tensorflow 1.14 foi escolhido por possuir uma boa documentação, comunidade ativa e código-fonte aberto. Este pacote possui o módulo Keras, uma API (Interface de Programação de Aplicativos) que possui camadas de neurônio já implementas e, principalmente, otimizadas. Desta forma pode-se desenvolver algoritmos de Deep Learning em alto nível de abstração.

Uma camada de neurônios é definida como uma operação do produto de tensores acrescida de uma operação não-linear aplicada a todos os elementos. Esta função é denominada função de ativação e tem como objetivo ponderar a passagem de cada elemento do tensor resultante do produto, de forma que o sinal pode ser amplificado ou atenuado para a próxima camada.

O modelo de rede gerado com o Keras será o algoritmo que processará os dados recebidos do *smartphone* e retorna o caractere identificado.

2.3 Revisão sobre sensores Android e iOS

Conforme a documentação dos sistemas operacionais iOS e Android (acessado em Dezembro de 2020), existem métodos prontos para qualquer aplicativo acessar as informações do acelerômetro do dispositivo. Em nenhum dos sistemas operacionais são feitas requisições para o usuário de permissão de acesso ao componente. Desta forma, uma aplicação pode monitorar o acelerômetro e captar vibrações de onde o dispositivo estiver posicionado.

Em nenhuma ocasião são feitas restrições explícitas para a taxa de amostragem, além das restrições inerentes ao sistema de preempção de processos de sistemas multi-tarefas. Conforme observado por Marquardt (2006), a frequência de amostragem máxima do iPhone 3GS é de aproximadamente 100 Hz. Portanto, pelo Teorema da amostragem de Nyquist–Shannon, podemos medir frequências mecânicas de até 50 Hz.

Estes dispositivos, majoritariamente, possuem microfone com largura de banda até 20 KHz, porém estes fazem requisição de acesso ao usuário em ambos sistemas operacionais. Com este mecanismo de segurança o usuário pode perceber que uma aplicação pode estar fazendo acesso indevido ao componente e não permitir o acesso.

Desta forma, os acelerômetros do dispositivo podem ser utilizados para ataques *Side-channel*, onde uma informação é obtida por um elemento terceiro ao alvo do ataque, neste caso o acelerômetro do celular em relação as teclas pressionadas em um teclado próximo.

2.4 Deep learning para analise de sons e vibrações

Deep Learning é uma técnica utilizada popularmente em classificação de imagens, visto que o conceito físico de entrada (as cores de cada pixel e sua posição) não é intuitivamente correlacionada com a classificação (o conteúdo a ser classificado na foto), porém já foi utilizado para análise de diferentes tipos de entrada por

Conneau (2016) e Lee et. al (2009) para classificação de texto e gêneros musicais, respectivamente.

Com o desenvolvimento de APIs de Deep Learning mais genéricas, podemos fazer classificadores de praticamente qualquer conjunto de dados de entrada, porém não há garantias de que o resultado será tão satisfatório quanto a classificação de uma imagem.

Desta forma, se pudermos utilizar uma representação em imagem de um determinado sinal, podemos utilizar o algoritmo de forma otimizada.

Isto pode ser obtido através do espectrograma do sinal, em que determinada entrada é transformada em um tensor de ordem 2, cujas dimensões são definidas pela resolução da frequência da FFT e pela quantidade de janelamentos para realizar as FFT.

Um espectrograma é uma representação bidimensional da amplitude de cada frequência de um sinal discreto. O sinal é janelado em pequenos períodos de tempos não sobrepostos e para cada janela é computado o módulo de sua FFT. Desta forma pode-se avaliar o comportamento da amplitude de cada uma das frequências do sinal ao longo do tempo.

Assim, obtemos um tensor semelhante a um tensor de uma imagem monocrática e este tensor é único para cada sinal, desde que sua frequência máxima seja menor que metade da frequência de Nyquist.

Marquardt utilizou pares de teclas pressionadas, pois observou que a identificação de teclas individuais tinha uma performance ruim quando comparado aos pares de teclas. Também se utilizou de um dicionario para identificar a em quais palavra cada par encontrado apareciam.

Berger utilizou aprendizado de máquina apenas com camadas de neurônios simples e obteve uma resposta que era muito sensível em função da quantidade de repetição das letras em cada palavra. Com isso, palavras com poucas ou nenhuma

repetição não conseguiam ser bem identificadas.

Na Iniciação científica prévia a este trabalho foi observado que a partir de 60 amostras por tecla são suficientes para classificar 9 teclas de um teclado. Também foi obtido uma configuração de camadas de neurônios através da resposta impulsiva de uma viga que será utilizada neste trabalho, pois a configuração se demonstrou capaz de classificar as teclas com poucas alterações que são especificadas na seção Procedimento Experimental.

2.5 Computação em nuvem

O instituto americano NIST (National Institute of Standards and Technology, 2011) define (em traduação livre) a computação em nuvem como um modelo para habilitar acesso online de recursos computacionais sob demanda, de forma conveniente, que pode ser provisionada e liberada com mínimo esforço de gerenciamento ou interação com o provedor do recurso. Por exemplo, um servidor dedicado a hospedar um banco de dados irá precisar de muito armazenamento e pouco processamento, e a quantidade necessária de armazenamento pode variar conforme a demanda. Um serviço de computação em nuvem (*cloud*) pode prover este servidor e garantir que ele estará disponível sempre que necessário e que seu espaço disponível seja "elástico", ou seja, possa aumentar conforme a demanda.

Conforme mais recursos são necessários, maior é o custo do serviço. Ainda assim, o custo tende a ser menor que a aquisição de um servidor dedicado, visto que não é necessário comprar um novo servidor caso as especificações do servidor sejam alteradas.

Neste trabalho foi utilizado um servidor *cloud*, pois é necessário que a plataforma esteja sempre disponível para receber os dados e que ela seja segura (no sentido de garantir que a porta aberta para a internet seja acessada apenas pelo aplicativo da plataforma).

O serviço utilizado foi a instância T2 micro do *Amazon EC2*, que possui 1 Gb de RAM e uma CPU, além de 10 Gb de armazenamento. Este serviço é inicialmente gratuito e após 12 meses é alugável a USD$\$0,0116$ por hora.

3 Procedimento Experimental

Foi preciso, antes de tudo, verificar se é possível fazer classificações de um sinal de vibração utilizando aprendizado de máquina para poder dar continuidade na inferência de teclas pressionadas, que adicionam outras variáveis ao problema de identificação. Então, foi elaborado um sistema de classificação com dados bem simplificados, como três pontos co-lineares em uma superfície plana.

Os procedimentos de aquisição serão adequados e embarcados em um *smartphone* e os procedimentos de processamento dos dados (treinamento da rede, inferência e avaliação) serão adequados a um servidor *cloud*.

3.1 Algoritmo de classificação de vibrações captadas por acelerômetro profissional

3.1.1 Dados obtido na superfície plana

Inicialmente foi marcado, em uma superfície plana, três pontos retilíneos distanciados de uma, três e oito polegadas do centro da superfície. No centro da superfície, e perpendicular a esta, foi fixado um acelerômetro unidimensional com taxa de amostragem de $20,48$ KHz.

Foi gerado um impulso mecânico pontual em um dos pontos e a aceleração captada pelo acelerômetro foi armazenada. Este processo se repetiu 10 vezes para cada um dos três pontos da superfície, totalizando 30 dados captados. Por fim, foi obtido o espectrograma de cada um dos dados com um janelamento de $30\mu s$ e frequência máxima de $10,24$ KHz, ou seja, metade da frequência de amostragem. Estes espectrogramas constituem o conjunto de amostras a ser utilizado.

Os espectrogramas foram utilizados para treinar uma rede neural profunda, implementada utilizando o pacote *Keras* do *Tensorflow*.

Diferentes sequências de camadas foram testadas de forma a otimizar o resultado final, assim como os hiper-parâmetros respectivos a cada camada. Também foram escolhidos diferentes *solvers* e funções de *loss*, também com o objetivo de otimização.

Por fim, o *solver* que demonstrou melhor acurácia final foi o "*Adam*", juntamente com a função de custo "entropia cruzada categórica". A métrica escolhida para avaliar o resultado foi a acurácia da identificação do conjunto de avaliação.

A rede foi treinada com 20 amostras e avaliada em outras 10. Para que o resultado não dependesse da amostragem, foi utilizado o K-fold e a acurácia média e seu desvio padrão foram anotados.

3.1.2 Dados amostrados em um teclado

Ao desenvolver uma plataforma móvel para a captação de dados do teclado, é necessário ter uma referência de como seria o processo de treinamento da rede em uma plataforma tradicional para estimar parâmetros de projeto, dentre eles:

- A quantidade de dados necessários para o treinamento;
- Quais as particularidades de algumas teclas;
- Qual a relação sinal-ruído;
- Quantas camadas são necessárias para a inferência (e, consequentemente, quanto processamento e memória são necessários);

Portanto foi desenvolvido um sistema de captação de dados do teclado utilizando um acelerômetro e sistema de aquisição profissional, seguido do treinamento de rede e uma aplicação prática da rede na inferência de senhas geradas aleatoriamente com os dados do conjunto de avaliação.

Captação dos dados Para a amostragem de um sinal real, foi utilizado um teclado mecânico da fabricante *Dell* conforme a Figura 2, um acelerômetro profissional triaxial *DeltaTron* com seu *hardware* para aquisição de dados, onde o sinal é amostrado e enviado um software, conforme as Figuras 3 e 4.

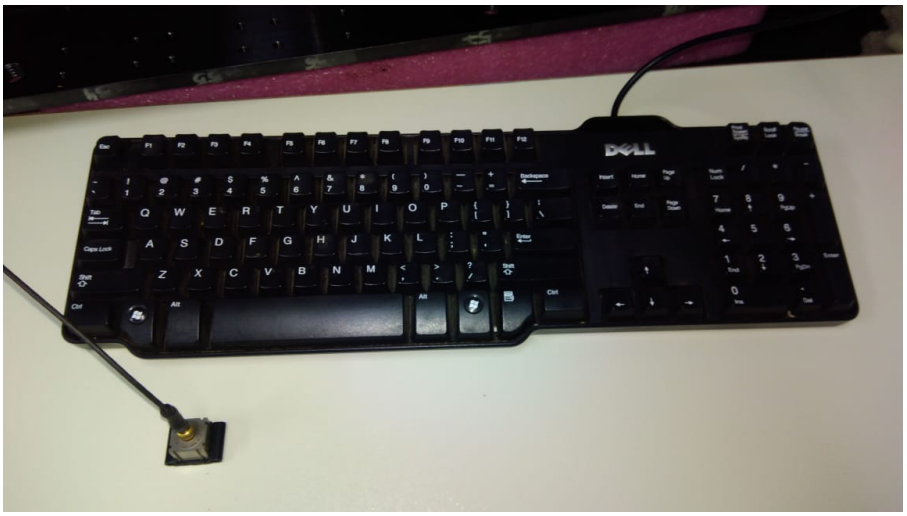

Figura 2: Teclado e acelerômetro na bancada de aquisição

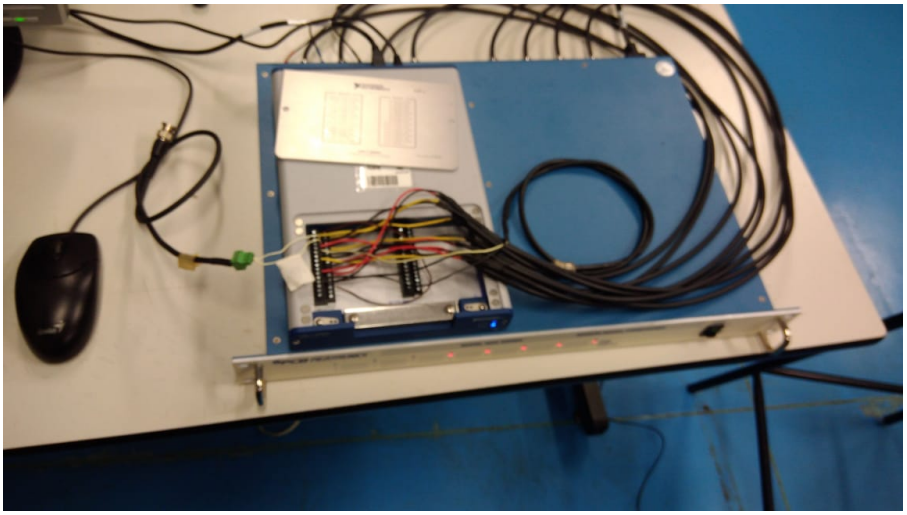

Figura 3: *Hardware* para aquisição do sinal do acelerômetro

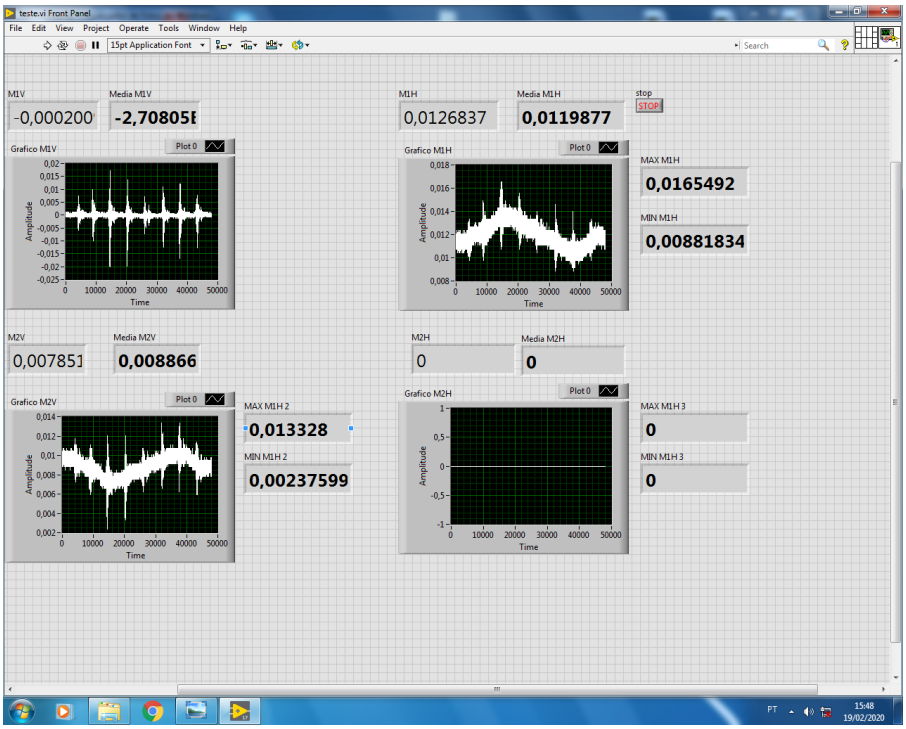

Figura 4: Software para aquisição do sinal do acelerômetro

O teclado é fixado à bancada utilizando uma fita dupla-face para que não deslize durante o experimento, sem influenciar nos resultados, e o acelerômetro é fixado utilizando uma cera para esta finalidade.

A captação dos sinais ocorreu apertando as teclas pausadamente, de forma a não sobrepor dois sinais, e com a mesma intensidade em que se utiliza o teclado usualmente. Como o acelerômetro é triaxial, pode-se captar as 3 dimensões da vibração e, inicialmente, as 3 seriam utilizadas para o treinamento da rede.

Dez teclas foram escolhidas de forma a mapear a parte alfabética do teclado, conforme a Figura 5. Os sinais da barra de espaços foram posteriormente descartados pois esta possui um mecanismo de três pontos de apoio, ao contrario das letras que possui apenas um. Logo a posição onde a tecla era apertada alterava o sinal significativamente.

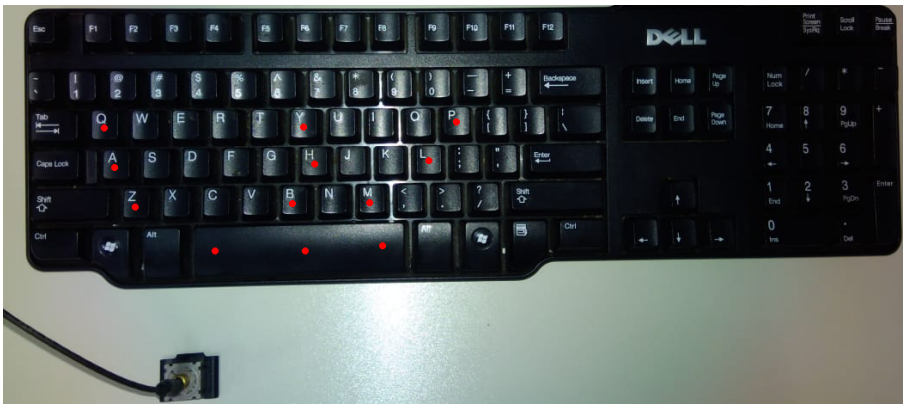

Figura 5: Pontos vermelhos indicam onde cada (e qual) tecla foi pressionada na captação dos dados

O sistema de aquisição fornecido registrou os sinais em bateladas de 10 pressionamentos por vez. Então esses sinais foram separados posteriormente utilizando um limiar de energia. A energia de um sinal discreto é calculado pela Equação 3 e, quando era encontrado um trecho (janelamento de 100 milissegundos conforme Berger e Marquardt [1] [2]) com energia 3 vezes maior que a energia do ruído, o trecho era separado com uma amostra de tecla.

$$E_s = \sum_{n=-\infty}^{+\infty} x^2[n] \tag{3}$$

Conforme será abordado na seção de Resultados, uma estimativa inicial seria de 60 amostras por categoria a ser classificada e foi utilizado 70 amostras por tecla como uma margem adicional.

Além da barra de espaços, as teclas escolhidas foram "q", "y", "p", "a", "g", "l", "z", "b"e "m".

Treinamento da rede Como um dos resultados da Iniciação Científica prévia era encontrar configurações de camadas para a análise de sinais de vibração, esta parte não precisou ser refeita para os dados amostrados no teclado. Entretanto, como as novas amostras possuíam dimensões maiores (por possuírem maior taxa de amostragem e, consequentemente, mais dados por amostra) e o banco de dados possuía 3

vezes mais categorias a serem determinadas (9 teclas contra 3 distâncias até o ponto de medição), a quantidade de camadas precisou ser reduzida para evitar *overfitting* (muitas camadas e muitas *features* por amostra contribuem mutualmente para o *overfitting*. A configuração resultante otimizada das camadas é apresentada na seção de Resultados Obtidos.

Assim como na pesquisa prévia à este Trabalho, foi necessário aleatorizar o banco de dados para diminuir a tendência de que cada sub-conjunto tenha mais ou menos amostras de alguma categoria e, consequentemente, o resultado final irá variar conforme a aleatorização. Portanto os testes foram repetidos até que a média entres as acurácias de cada execução convergisse para algum valor. A mesma proporção de dados foi utilizada para os sub-conjuntos: 70% para treinamento, 20% para validação cruzada e 10% para avaliação.

A priori espera-se que ao utilizar as 3 dimensões do acelerômetro a acurácia seria maior que utilizar apenas a dimensão principal (paralela ao movimento da tecla). Portanto foram feitos testes com as duas configurações e seus resultados foram anotados.

Sobreposição de sinais Um dos fatores relevantes para a composição de uma senha é a interferência do sinal de uma tecla na próxima, então foi feito um estudo sobre como a acurácia média de cada tecla individual é afetada com a sobreposição de outro sinal sobre o grupo de avaliação.

A quantidade de do sinal sobreposta foi variada de 1% até 100%, onde este valor representa a proporção de sinal anterior somado aos sinais do grupo de avaliação.

Os dados foram coletados e serão apresentados no Relatório Final deste Trabalho de Graduação.

3.1.3 Identificação de senhas

Como os classificadores possuem diferentes acurácias conforme a aleatorização, foram gerados 10 classificadores e 130 senhas aleatórias com as letras utilizadas no grupo de avaliação, divididas em 13 grupos de 10 onde, em cada grupo, a senha possui n caracteres, com $1 <= n <= 13$.

Em uma iteração do algoritmo de identificação de senhas é selecionado uma amostra do sub-conjunto de avaliação para cada caractere de uma senha e os une, conforme a Figura 6 para a senha gerada "zgqm".

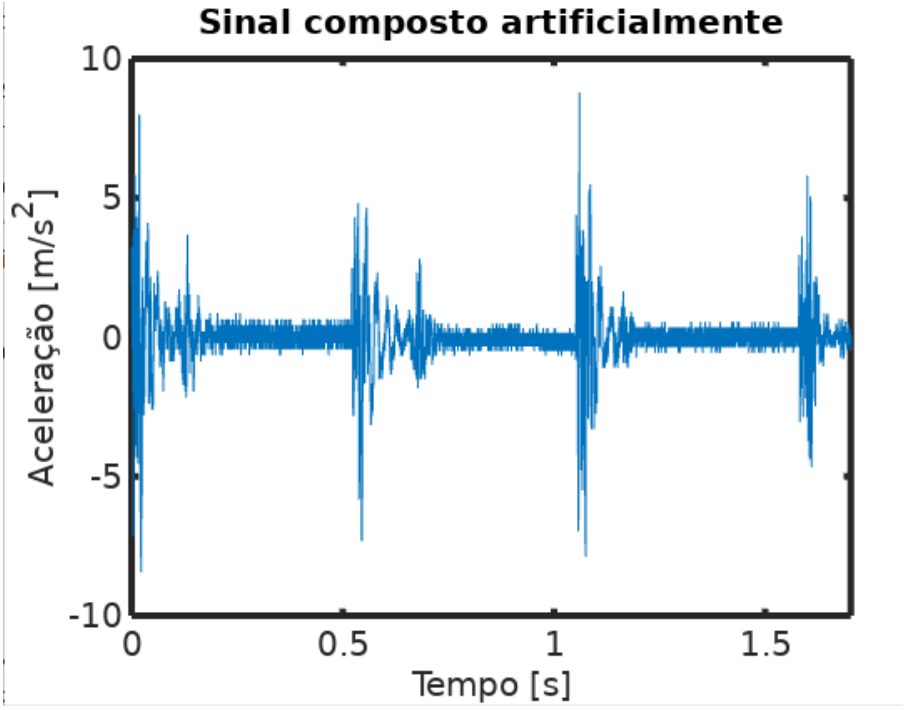

Figura 6: Sinal composto para a senha *zgqm*

O sinal composto artificialmente então passa pelo algoritmo de avaliação, onde é separado em n caracteres e classificados pela rede. Os resultados de sucesso são anotados juntamente da quantidade de teclas e a compilação é apresentada na seção Resultados Obtidos.

3.2 Arquitetura da Plataforma

Com o Algoritmo de classificação de vibrações operacional, a plataforma pode ser desenvolvida para captar os dados do *smartphone* e efetuar a classificação dos sinais de vibração.

Para isso, o problema foi separado nas seguintes etapas:

- Captar a vibração do acelerômetro de um *smartphone*
- Enviar os dados para o algoritmo de classificação
- Retornar a classificação para o dispositivo

A aquisição dos dados pode ser feita através de um aplicativo executado no *smartphone*, entretanto, o algoritmo de classificação exige bastante recurso, o que não é desejado em um sistema embarcado. Além disso, o código desenvolvido em *python* precisaria ser convertido em um binário Java para ser executado no sistema Android e isso pode ser bastante complexo.

Portanto a plataforma proposta possui além de um aplicativo, um servidor *cloud* para receber os dados coletados, executar a classificação e retornar a classificação ao *smartphone* .

3.3 Cloud

A utilização de um serviço em nuvem é mais adequada à plataforma, visto que ela precisa estar sempre disponível (já que o aplicativo pode ser executado a qualquer momento) e deve consumir poucos recurso (como será apresentado na seção de Resultados e Discussões).

Foi utilizado um servidor EC2 da *Amazon Web Services - AWS*, configurado para receber dados apenas do IP da rede do *smartphone* com o aplicativo (evitando

ataques de negação de serviço).

Neste servidor foi programado um serviço (em Node.js) que recebe requisições HTTP do tipo POST com uma lista de sinais de aceleração, no formato "[horário Unix, aceleração no eixo x, aceleração no eixo y, aceleração no eixo z]"e os dados recebidos são armazenados em um arquivo no formato ".csv". Este formato foi escolhido para garantir que o "Serviço Loop"(explicado posteriormente) tenha facilidade na leitura dos dados em troca de um consumo maior de armazenamento.

O serviço também atende requisições do tipo GET, em que emite o conteúdo de um arquivo de texto que é constantemente atualizado pelo "Serviço Loop".

3.3.1 Serviço *Loop*

O "Serivço Loop"tem o papel de interpretar os dados coletados pelo *smartphone* e retornar uma classificação.

Nesta aplicação o serviço se utiliza da vibração captada ao pressionar uma tecla em um teclado e retorna a tecla pressionada, com o algoritmo gerado na subseção 3.1 deste Trabalho.

Entretanto, o serviço pode ser alterado para qualquer outro fim que respeite a estrutura de ler os dados no formato ".csv"e retorne dados em formato texto, tornando a plataforma útil para diversas aplicações.

3.3.2 Organização geral da plataforma

Como as etapas de desenvolvimento do modelo de *Machine Learning* e do desenvolvimento do Aplicativo exigem recursos da máquina, estes devem ser desenvolvidos em uma máquina local e não no servidor *cloud*.

A configuração da plataforma é apresentada na Figura 7.

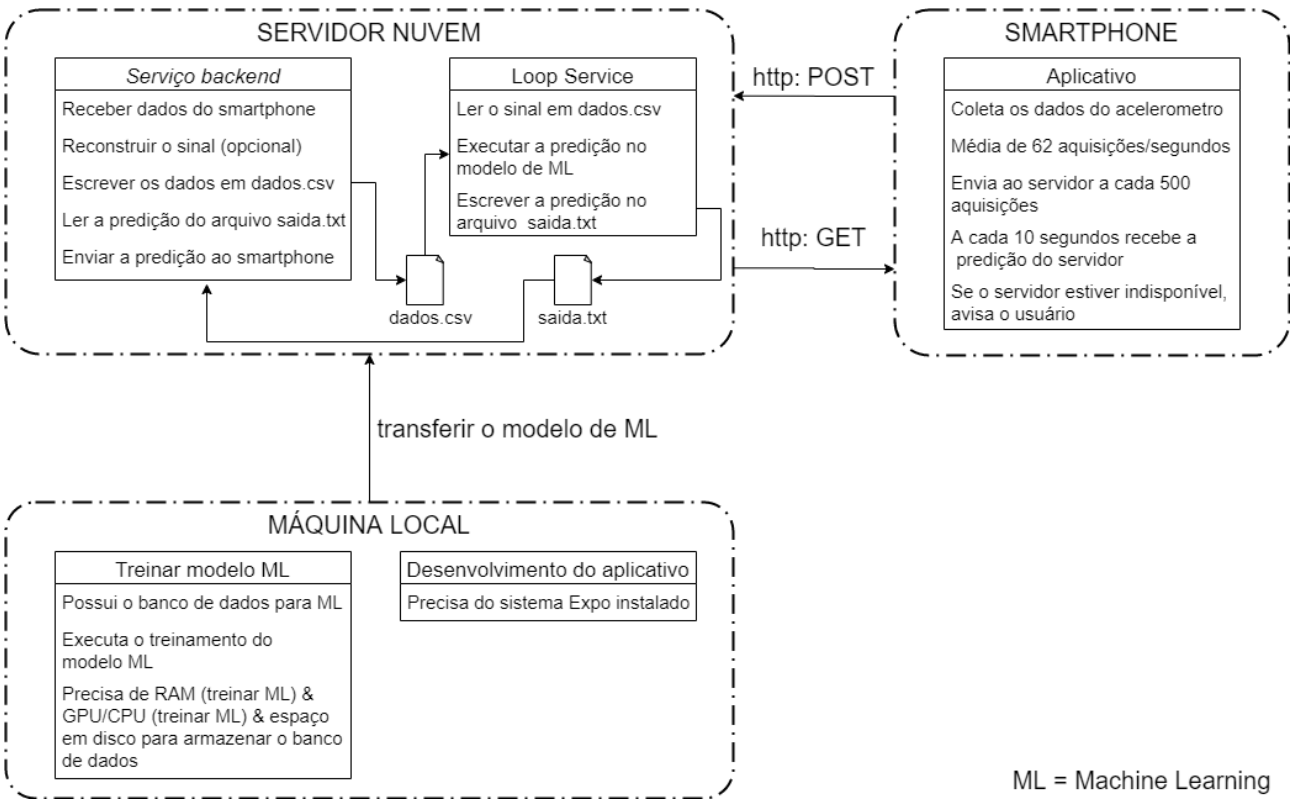

Figura 7: Diagrama da plataforma proposta

3.4 Aplicativo

Utilizando o *framework Expo* foi gerado um aplicativo para captar a vibração do teclado através do acelerômetro de um *smartphone*. Este framework permite gerar aplicativos para Android e iOS com o mesmo código-fonte.

O aplicativo faz a aquisição na maior taxa de amostragem possível e envia os dados em bateladas de 500 amostras para o servidor *cloud* pelo método HTTP POST.

O valor de 500 amostras foi obtido para minimizar o *overhead* da chamada do método POST, que seria significativa ao enviar uma amostra de cada vez.

Simultaneamente o aplicativo faz requisições GET ao servidor para receber a classificação da tecla pressionada.

O aplicativo também foi desenvolvido de forma a alterar o IP do servidor dina-

micamente, de forma a não ser necessário gerar um novo binário ao trocar o endereço do servidor *Cloud*.

3.5 Adequação do sinal do celular à rede treinada

Como a rede foi treinada com um acelerômetro e sistema de aquisição profissionais, os dados possuem uma alta taxa de amostragem e resolução. Este não é o caso dos dados provenientes do celular, logo foi necessário adequá-los à rede.

Assumindo que a taxa de amostragem do *smartphone*, mesmo sendo menor que o acelerômetro profissional, é maior que o dobro da maior frequência mecânica relevante à rede, podemos reconstruir o sinal amostrado no tempo e re-amostra-lo na frequência utilizada no treinamento da rede. Pelo Teorema da Amostragem, o sinal contínuo $x(t)$ que foi amostrado em um sinal discreto $x_a[n]$, a uma taxa $f_s = 1/T$, é dado pela equação 4, desde que a maior frequência de $x(t)$ seja menor que $f_s/2$.

$$x(t) = \sum_{n=-\infty}^{+\infty} x_a[nT] \frac{sen[\frac{\pi}{T}(t - nT)]}{\frac{\pi}{T}(t - nT)} \tag{4}$$

Para obter um sinal $x_{a_2}[n]$ amostrado na frequência de amostragem $f_{s_{orig}}$ em que a rede foi treinada, basta utilizar equação de amostragem 5.

$$x_{a_2}[n] = x(n/f_{s_{orig}}) \tag{5}$$

Ainda assim, a frequência $f_s = 1/T$ precisa ser constante o que pode ser obtido em um equipamento profissional, mas não é a realidade de um *smartphone* .

Com um *kernel* Linux, sobreposto a um sistema operacional Android em uma máquina virtual Java, onde vários aplicativos e serviços são executados em paralelo, não há como garantir que existirá uma constância na taxa de aquisição sem a

utilização de atrasos para sincronização, o que reduziria a taxa de amostragem.

Logo, foi necessário verificar qual era a variação média da taxa de amostragem e desenvolver um estudo para identificar o quanto essa variação compromete a reconstrução do sinal.

3.5.1 Reconstrução de sinal com taxa de amostragem variável

Utilizando a plataforma MATLAB, foi simulado um sinal composto de duas senoides de $6Hz$ e $15Hz$, simulando uma vibração mecânica. Os sinais não foram amortecidos para garantir que a frequência de Nyquist teórica seja finita e neste sentido diferem de um sinal de vibração do teclado.

Este sinal foi então amostrado em 62 Hz e posteriormente reconstruído, conforme a equação 4 e a diferença entre o sinal original e a reconstrução foi anotado.

Em seguida o mesmo sinal foi re-amostrado com uma frequência de amostragem variável $62 \pm \Delta$ Hz, onde $1 < \Delta < 47$ Hz, $\Delta \in \mathbb{N}$, reconstruídos conforme a Equação 4 e o erro (subtração entre o sinal original e a reconstrução) foi armazenado.

Então para cada erro foi calculado a relação sinal/erro de reconstrução (denotada por SER - *Signal-Error ratio*) , em função da variação da frequência à frequência média. Este método é calculado conforme a equação 6, similar a utilizada para calcular a relação sinal/ruído (SNR). Para as magnitudes M do sinal e do erro foi utilizado o valor RMS do sinal e do erro.

$$SER = 10.log_{10}(\frac{M_{\text{sinal}}}{M_{\text{ruído}}}) \tag{6}$$

Esta métrica foi utilizada apenas para comparar a amplitude do erro em relação ao sinal em decibéis e não possuí nenhuma relação com a relação sinal/ruído além da similaridade das fórmulas.

O valor médio de amostragem de 62 Hz foi escolhido com a informação a *posteriori* que a taxa de amostragem média do aplicativo é de 62 Hz.

O valor máximo de variação utilizado foi 47 Hz foi escolhido com informação obtida a *posteriori* em que valores maiores possuíam $SER > 0$, ou seja, o valor RMS do erro era maior que o valor RMS do sinal e a reconstrução já não possuí relevância. Como a maior frequência do sinal é de $15Hz$, a taxa de amostragem deve ser de, no mínimo, $30Hz$ para a reconstrução, conforme o teorema de Nyquist. A taxa de amostragem média foi fixada em 62 Hz, mas com uma varição de 47 Hz já teríamos o intervalo $15 - 109$ Hz e a aquisição fica muito comprometida. Logo aquisições com variações ainda mais altas já violam as hipóteses da reconstrução do sinal e não foram mensuradas.

3.5.2 Amostragem e reconstrução de sinal captado pelo celular

Em seguida o sinal do *smartphone* foi captado e reconstruído utilizando a mesma técnica para obter os dados na mesma taxa de amostragem utilizada no modelo de *Deep Learning*

4 Resultados e Discussões

4.1 Algoritmo de classificação de vibrações captadas por acelerômetro profissional

4.1.1 Dados obtido na superfície plana

As Figuras 8 e 9 representam respectivamente o sinal e o espectrograma de uma amostra a uma polegada do sensor. As Figuras 10 e 11 representam respectivamente o sinal e o espectrograma de uma amostra a oito polegadas do sensor.

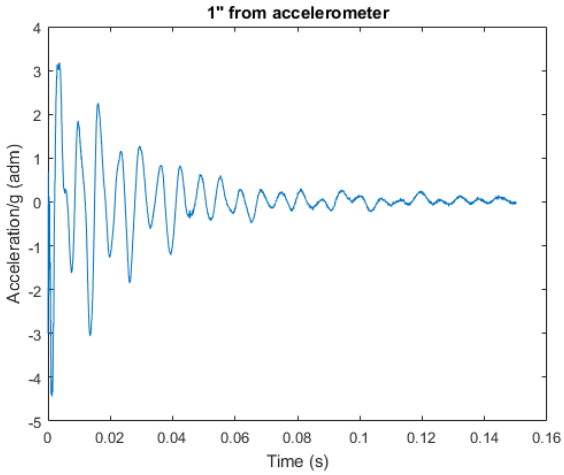

Figura 8: Sinal captado pelo acelerômetro com impulso a 1 polegada do sensor

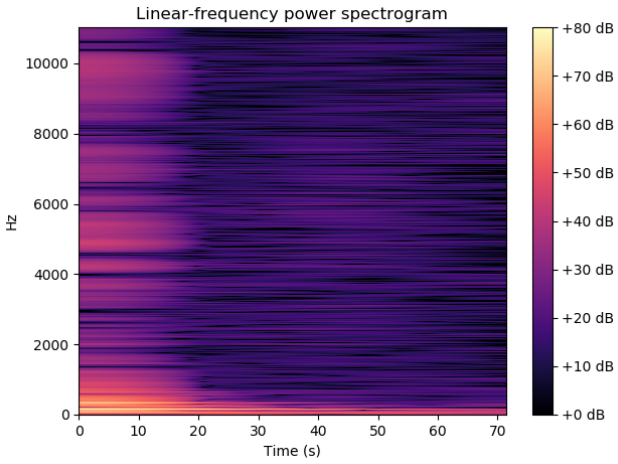

Figura 9: Espectrograma do sinal representado na figura 8

Observou-se que as harmônicas dos dados próximos ao sensor estão mais agrupadas, enquanto as harmônicas do sinal mais distante estão mais dispersas e em maior quantidade. Também foi possível notar que com o passar do tempo o ruído fica muito significativo em relação ao sinal e, portanto, as ultimas colunas dos espectrogramas não tendem a agrupar, independentemente da distância entre o impulso e o sensor. Entretanto, as ultimas colunas ainda podem ser distinguida do sinal afastado do sensor pela diferença da intensidade média das harmônica, que é menor no sinal próximo ao sensor.

Com isso pode-se supor que uma boa hipótese inicial para classificação das

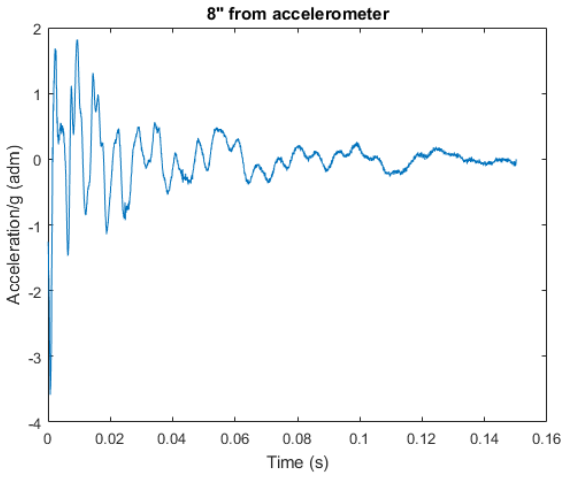

Figura 10: Sinal captado pelo acelerômetro com impulso a 8 polegadas do sensor

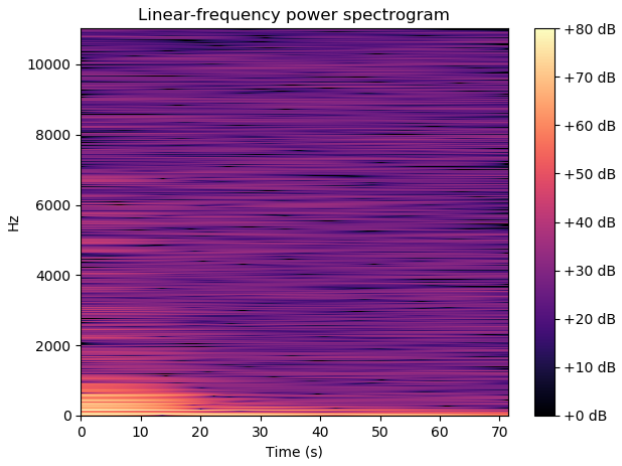

Figura 11: Espectrograma do sinal representado na figura 10

distâncias é a identificação dos agrupamentos (*cluster*) das primeiras colunas de cada espectrograma. Porém, deve-se ressaltar que o objetivo do *Deep Learning* é que a própria combinação de camadas possa inferir e ponderar quais características são relevantes, através dos otimizadores e da função de custo. Portanto não foram feitas modificações nos dados de forma a facilitar que esta hipótese fosse utilizada pela rede.

Os dados de entrada que compõe o conjunto de amostras foram treinados conforme a seção "Procedimento Experimental". Foi observado que o desvio padrão da acurácia pelo K-fold era significativamente alto e a acurácia baixa, mais especificamente, uma acurácia média de 55% com 10% de desvio-padrão. Isto implica que o modelo não seria confiável em uma aplicação real, visto que ele poderia facilmente convergir para um modelo ruim, dependendo dos dados de avaliação.

Uma baixa acurácia implica que a rede não estava identificando quais as melhores características do espectrograma que diferenciavam as distâncias, mas apenas encontrava um modelo bom para qualquer conjunto de treinamento.

Uma possível explicação para a inconsistência da acurácia é que ela é afetada pela aleatoriedade da separação dos conjunto, de forma que algumas amostras não seriam identificáveis em um cenário real. Ela também pode ser afetada pela alea-

toriedade dos pesos iniciais da matriz, de forma que a rede não conseguiu treinar o modelo utilizando apenas características do conjunto de amostras, mas acabou se beneficiando da aleatoriedade dos pesos em algumas execuções.

Foi observado que em alguns casos o modelo conseguia treinar todo o sub-conjunto de treinamento, mas falhava no sub-conjunto de avaliação. Isso é conhecido com sobre-treinamento, ou *overfitting*, onde a rede encontrou um modelo que não identifica a característica desejada, mesmo que o *loss* tenda a diminuir no sub-conjunto de treinamento. Neste caso a rede conseguiria identificar apenas as amostras treinadas.

Para reduzir o *overfitting*, a quantidade de filtros por camada de convolução foi reduzida a um filtro de dimensão 3x3. Neste cenário alguns resultados ainda tinham *overfitting*, embora a maioria não conseguiu convergir para um modelo adequado, mesmo para o sub-conjunto de treinamento. Neste caso, o modelo estava sub-treinado (conhecido como *underfitting*). As figuras 12 e 13 mostram o *loss* para o sub-conjunto de treinamento e validação ao longo das épocas de treinamento para as situações de *underfitting* e *overfittng*, respectivamente.

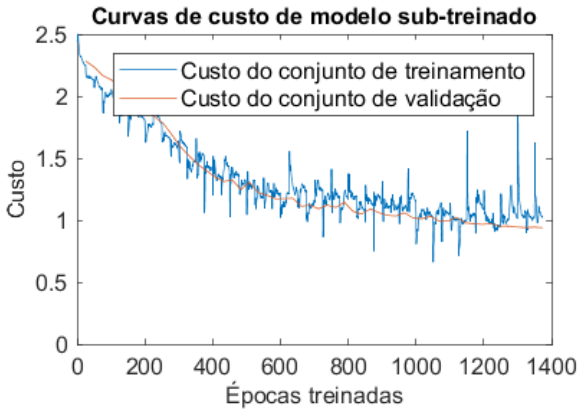

Figura 12: Custo ao longo das épocas de treinamento com modelo sub-treinado

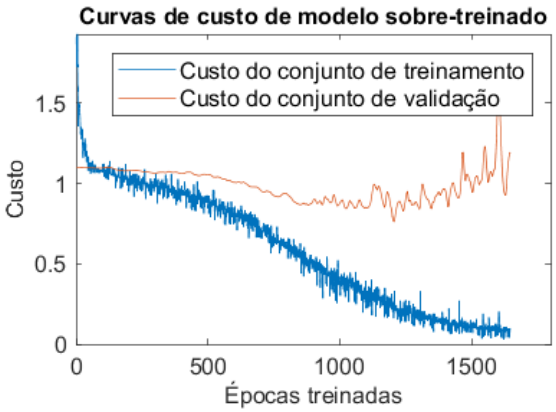

Figura 13: Custo ao longo das épocas de treinamento com modelo sobre-treinado

Podemos observar pela tendência do gráfico de *underfitting*, o erro não diminui significativamente com o aumento das épocas. Provavelmente o algoritmo convergiu para um mínimo local, diferente do mínimo global esperado, e não foi capaz de obter um modelo que mapeasse o conjunto de treinamento para suas respectivas

classificações.

No gráfico de *overfitting* podemos observar que as curvas dos sub-conjuntos de treinamento e validação se separam, de forma que o conjunto de treinamento tende a ser perfeitamente classificado, mas uma nova amostra não seria classificada com precisão.

4.1.2 Resultados da aquisição do teclado

Um exemplo do sinal captado é apresentado na Figura 14. Pode-se observar que existem dois picos dignificativos no sinal. O primeiro é causado ao pressionar a tecla, o segundo acontece quando a tecla retorna a sua posição original pelo sistemas de molas do teclado.

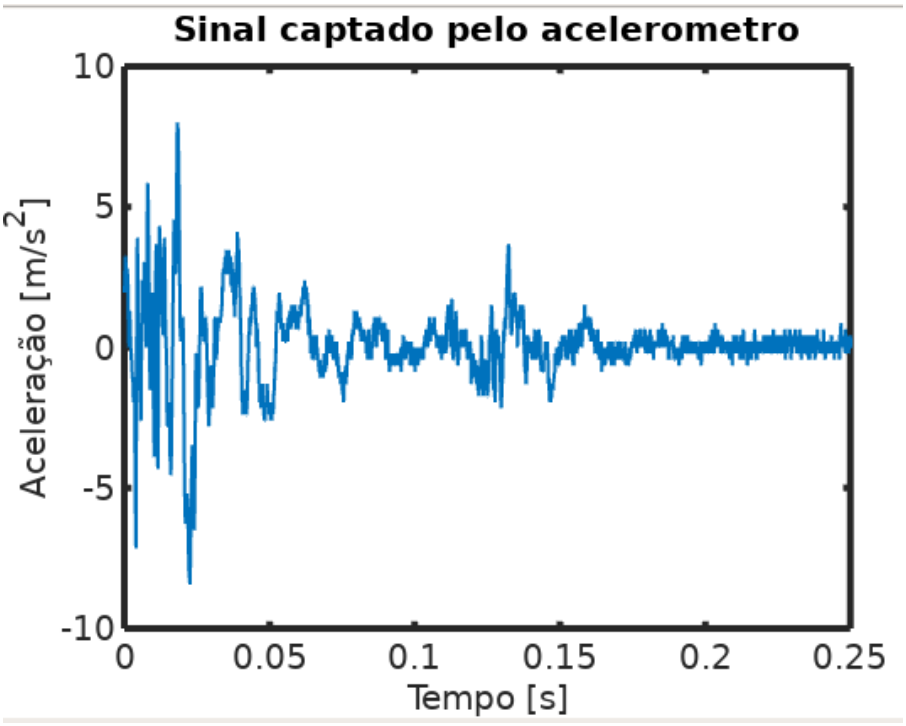

Figura 14: Sinal captado pelo acelerômetro após pressionar uma tecla no eixo paralelo ao movimento da tecla

A relação de amplitude entre os picos é característica em cada teclada, seja por diferenças nos sistemas mecânicos, seja pela dissipação de cada sinal até que ele chegue ao acelerômetro.

Utilizando o janelamento Hann e o menor número de amostras entre colunas de *FFTs* adjacentes, obtemos uma matriz com a maior resolução possível no tempo e na frequência. Entretanto matrizes grandes são custosas para operar e serão posteriormente manipuladas por uma camada de *Pooling*. Desta forma pode-se normalizar sinais com variados tempos totais de sinal e com diferentes taxas de aquisição. Isto é necessário visto que teclas mais afastadas do acelerômetro dissipam mais rápido e para a aquisição em um *smartphone* a taxa de amostragem será menor.

O conjunto de camadas utilizado foi o seguinte:

- MaxPooling2D: Janelamento em função da dimensão da entrada de forma que o tensor resultante seja de 100x100
- BatchNormalization;
- Conv2D: 48 filtros de tamanho $4x4$
- Dropout: 25%, mantendo 75% da camada anterior
- Flatten;
- Dense: ativação *softmax* e quantidade de neurônios igual a quantidade de teclas treinadas (9 neste experimento)

Este foi o conjunto mais otimizado para o sinal de vibração do teclado. Desta-se que a quantidade de camadas é menor que a utilizada com os sinais simulados (da pesquisa com viga na Iniciação Científica) e com menos neurônios por camada. Isso se deve a alta tendência de *overfitting* da matriz por ter dimensões maiores que as obtidas no sinal simulado.

Para os sinais compostos das três dimensões, a acurácia média do grupo de avaliação foi de 49% com desvio padrão de 5%. Para os sinais de uma dimensão (paralela ao movimento da tecla), a acurácia foi de 70% com desvio padrão de 6%.

Este resultado mostra que, embora a utilização de um sinal tridimensional possa

distinguir ondas normais e cisalhante, o aumento na quantidade de *features* por amostra torna mais difícil para o algoritmo convergir a um resultado ótimo.

A curva de custo média por épocas é apresentada na Figura 15. Observa-se que, embora o resultado tenha sido suficiente para classificar as teclas com uma boa acurácia, o sistema ainda está sobre-ajustado, pois a curva do grupo de validação estagnou enquanto a do grupo de treinamento ainda decaia. Como não foi encontrada nenhuma combinação de camadas que otimizasse esse resultado, pode-se supor que para reduz ainda mais o erro seria necessário um *dataset* ainda maior.

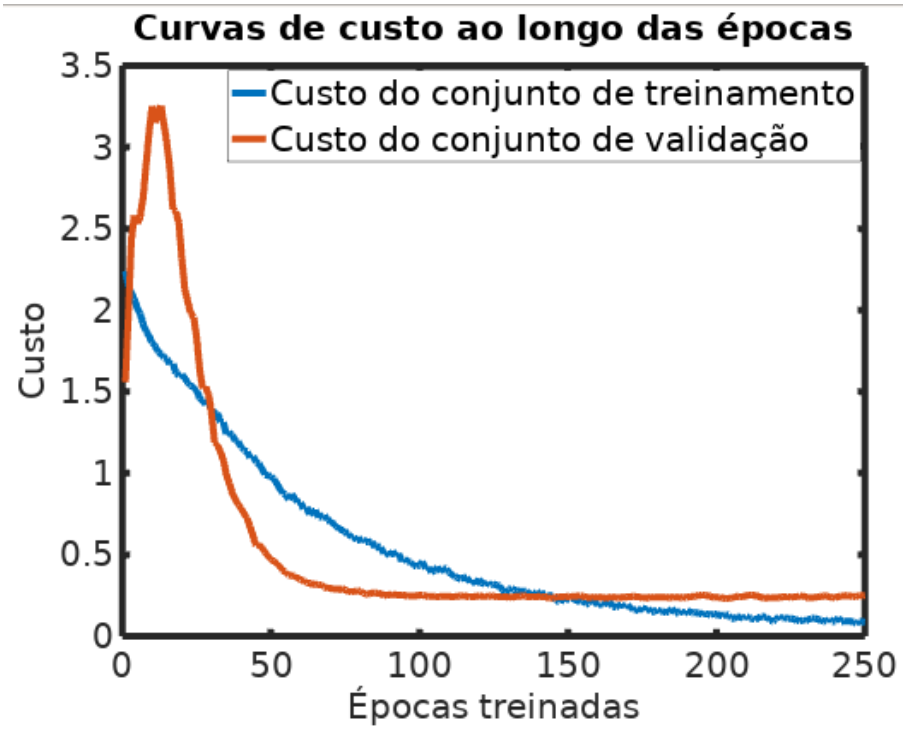

Figura 15: Erro (ou custo) de dos sub-conjuntos de treinamento e validação em função das épocas

4.1.3 Identificação de senhas

Para o conjunto gerado de 130 senhas de 1 até 13 caracteres, foi observado que há um decaimento exponencial conforme a quantidade de caracteres na senha. A Figura 16 apresenta a acurácia das 130 combinações de senhas (de 1 até 13 caracteres) para 10 retreinamentos da rede. A melhor curva exponencial que aproxima o modelo possui coeficiente de determinação (conhecido como R^2) igual a 0.993 e equação

$y = 1.01 * e^(- 0.363 * n)$ onde y é a acurácia e n a quantidade de caracteres.

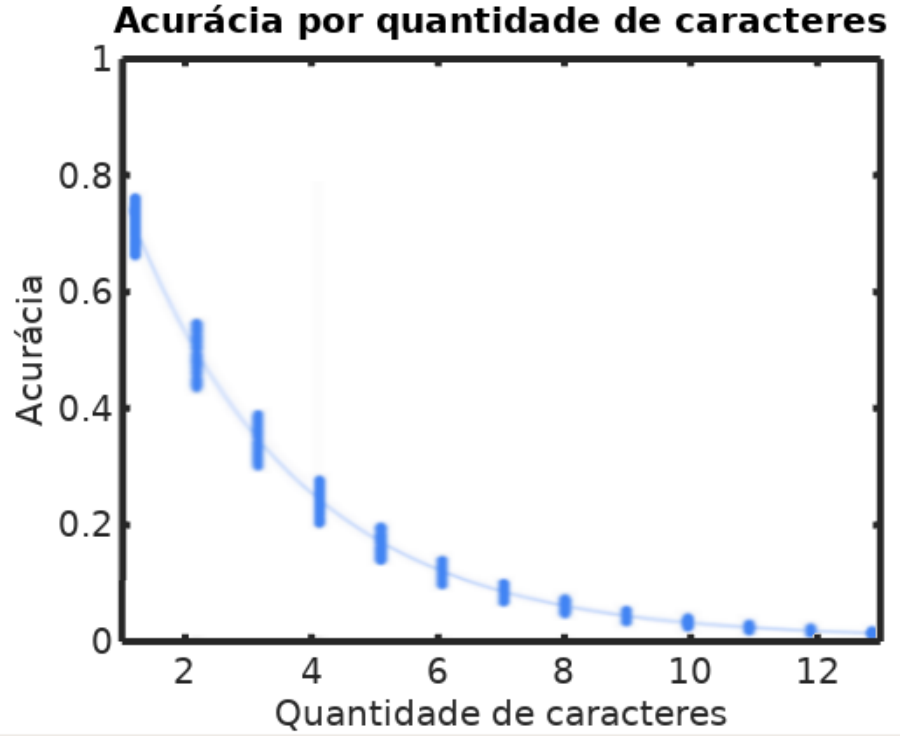

Figura 16: Acurácia por quantidade de caracteres em uma senha gerada aleatoriamente

Portanto, embora a acurácia para identificar cada tecla individualmente tenha sido muito boa, ao tentar identificar todas as teclas pressionadas em uma senha de 6 caracteres o algoritmo irá acertar a senha em 11.4% dos casos. Para fins de comparação, a chance de acertar uma senha de 6 caracteres (formadas pelas mesmas 9 letras do banco de dados) de forma aleatória é de 0.0002%.

4.2 Cloud

O serviço de Cloud se demonstrou estável durante todo o desenvolvimento do projeto. Não houve interrupção do servidor em nenhum dos momentos de aquisição.

4.2.1 Armazenamento

O serviço de armazenamento de dados de apenas $10Gb$ é restritivo apenas num cenário onde o algoritmo de aprendizado de máquina será treinado, e por isso esta etapa é realizada em uma máquina externa, conforme o diagrama apresentado na seção Procedimento Experimental.

Ainda assim, o armazenamento foi suficiente para instalar todas as dependências do *Node.js*, *TensorFlow/Keras* (incluindo o binário de um modelo treinado), sistema operacional e manteve $5Gb$ disponíveis para os dados do *smartphone* .

Cada dado recebido do *smartphone* possui 74 bytes no formato texto e, assumindo uma taxa de amostragem média de 62 amostras por segundo (como será apresentado na subseção "Aplicativo"), obtemos um tempo maior que 12 dias de aquisições contínuas, ou seja, não há necessidade de comprimir os dados por motivos de armazenamento escasso.

Assumindo que durante um treinamento de rede seja pressionado uma tecla por segundo, a rede suportaria 1.0610^6 amostras, mais de 1700 vezes a quantidade utilizada no treinamento da iniciação científica prévia a este trabalho, demonstrando que essa solução é robusta para resolver os problemas de *overfitting* por poucas amostras na questão de armazenamento.

Como já abordado, não é recomendável que o servidor cloud seja usado para o treinamento, mas este pode redirecionar os dados para a máquina em que este será feito.

4.2.2 Processamento e memória

O servidor não realizou treinamento da rede, logo não se esperava da máquina muito processamento e a escolha do servidor privilegiou custo ante processamento. Ainda assim este foi capaz de receber os dados do *smartphone* simultaneamente

a execução de avaliação do algoritmo de *Machine Learning* previamente treinado. Como essa é a situação máxima de estresse do servidor, ele supre todas as necessidades de processamento.

Quanto a utilização da memória RAM, restrita a apenas $1Gb$, este é capaz de manter o sistema operacional em paralelo a execução do *Node.js* com apenas 200 Mb, garantindo espaço suficiente para os momentos de pico do algoritmo de avalição que precisa carregar o binário na memória.

4.2.3 Segurança

Após a configuração inicial não houve nenhum incidente de segurança no servidor. De fato, foi apenas possível acessar os dados pela lista de IPs fornecidos, garantindo que apenas o aplicativo pudesse acessa-lo.

Diversos sistemas utilizam a suíte de serviços AWS e isto torna qualquer incidente de segurança suscetível a afetar serviços importante e, consequentemente, possui equipes diversas garantindo a segurança dos serviços. Esta centralização é uma vantagem em relação a desenvolver seu próprio servidor (ao invés de utilizar um serviço de *cloud*), onde é necessário manter suas próprias medidas de segurança.

4.3 Aplicativo

O aplicativo desenvolvido (apresentado na Figura 17) é capaz de enviar os dados do acelerômetro com uma frequência de amostragem média de 62 amostras por segundo, com uma varição ± 2 amostras por segundo.

Este valor é inferior ao observador por Marquardt[2] pois a plataforma de desenvolvimento Expo possui camadas de abstrações para transformar o mesmo código-fonte em binários diferentes para Android e iOS. Estas camadas de abstrações implicam em uma maior pilha de chamadas de processos até o *hardware* e, consequentemente, em atraso e a queda na taxa de amostragem.

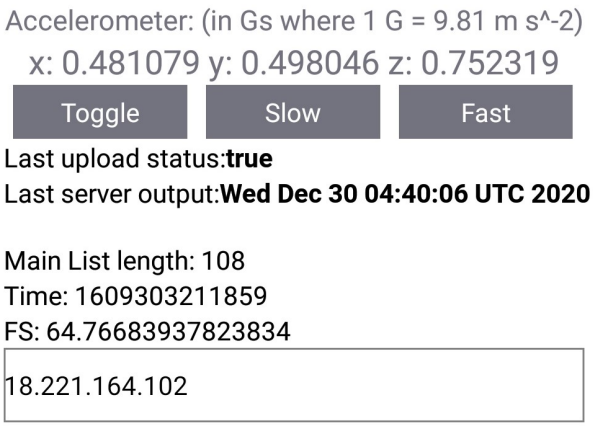

Figura 17: Tela do aplicativo

Assumindo que frequência de amostragem mínima seja de $60Hz$, podemos reconstruir sinais de aceleração de até $30Hz$, conforme o teorema da amostragem de Nyquist-Shannon.

O três eixos podem ser capturados e a Figura 18 apresenta a captura destes dados ao se pressionar uma tecla em um teclado próximo por 6 vezes.

Pode-se observar que o eixo "z"possui os sinais bem distintos do ruído, o que não é verdade para os outros dois eixos. Isto se deve ao fato de que o eixo "z"é paralelo ao movimento da tecla no teclado e, conforme observado na Iniciação Científica, é onde o sinal do teclado libera a maior parte da energia mecânica.

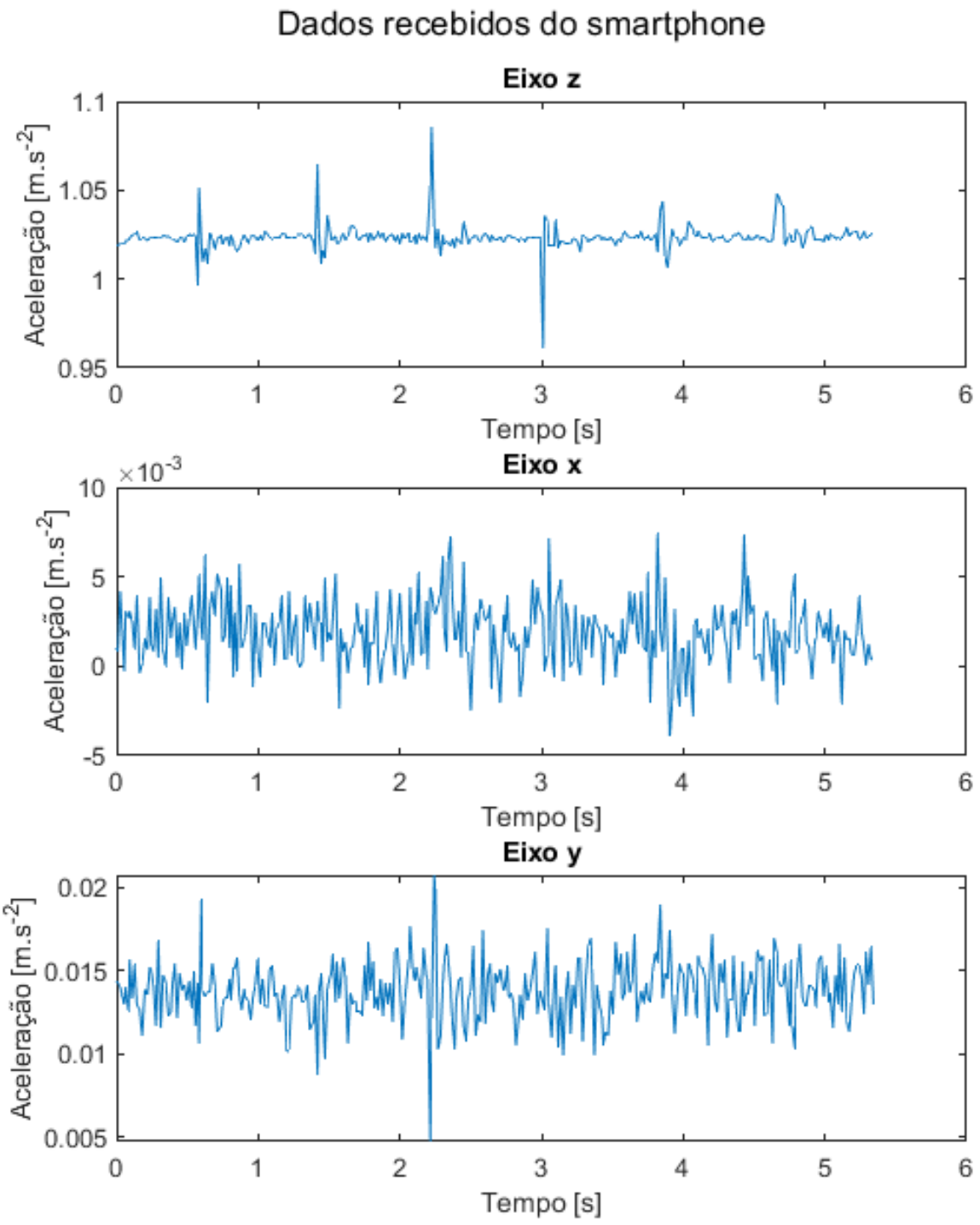

Figura 18: Dados de aceleração recebidos do *smartphone* no servidor

4.4 Adequação do sinal do celular à rede treinada

Os resultados dos experimentos de reconstrução de sinal são apresentados na seguinte forma: A figura 19 apresenta um sinal e sua amostragem a 62 Hz. A figura 20 apresenta o erro da reconstrução.

Ao variar a frequência de amostragem, como acontece no *smartphone*, obtemos a figura 21 e, por fim, a figura 22 apresenta a relacão do erro ao sinal em decibéis.

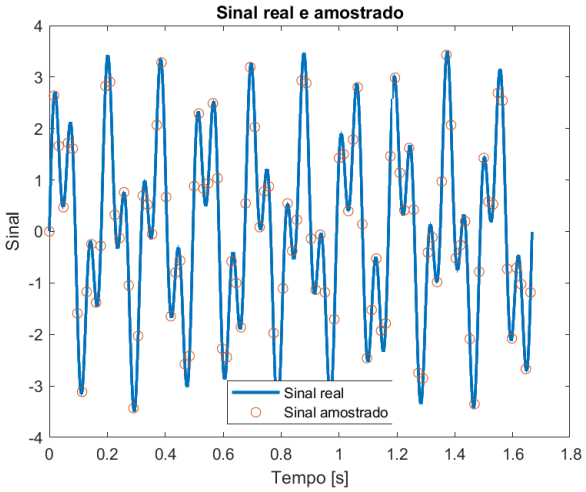

Figura 19: Sinal real e amostrado em uma frequência constante

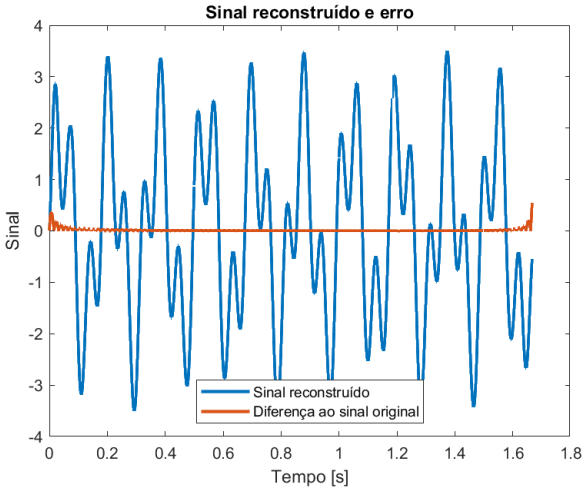

Figura 20: Sinal reconstruído (a partir de amostragem com frequência constante) e erro em relação ao sinal real

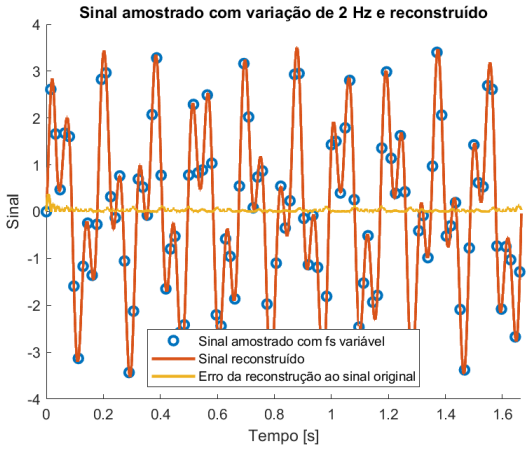

Figura 21: Sinal amostrado (com frequência de ± 2 Hz), reconstrução e erro em relação ao sinal real

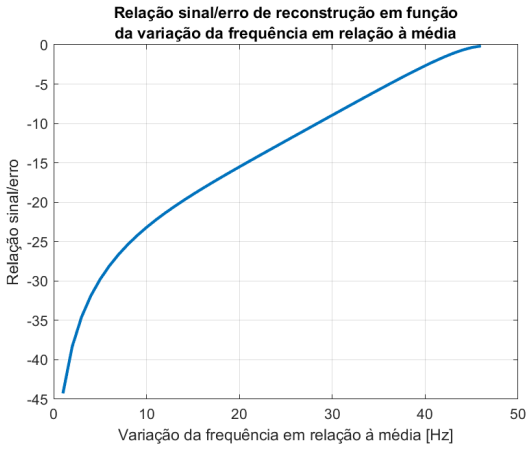

Figura 22: Relação sinal erro em decibéis em função da variação do desvio frequência de amostragem à média

Como apresentado na seção correspondente ao aplicativo, o desvio da frequência é de apenas $2Hz$, logo o erro do sinal reconstruído em relação ao sinal verdadeiro é desprezível (-40 dB), se assumirmos que a frequência média de aquisição for maior que a frequência de Nyquist.

4.4.1 Amostragem e reconstrução de sinal captado pelo celular

O sinal de vibração captado pelo *smartphone* assim como a reconstrução do seu sinal são apresentados na Figura 23.

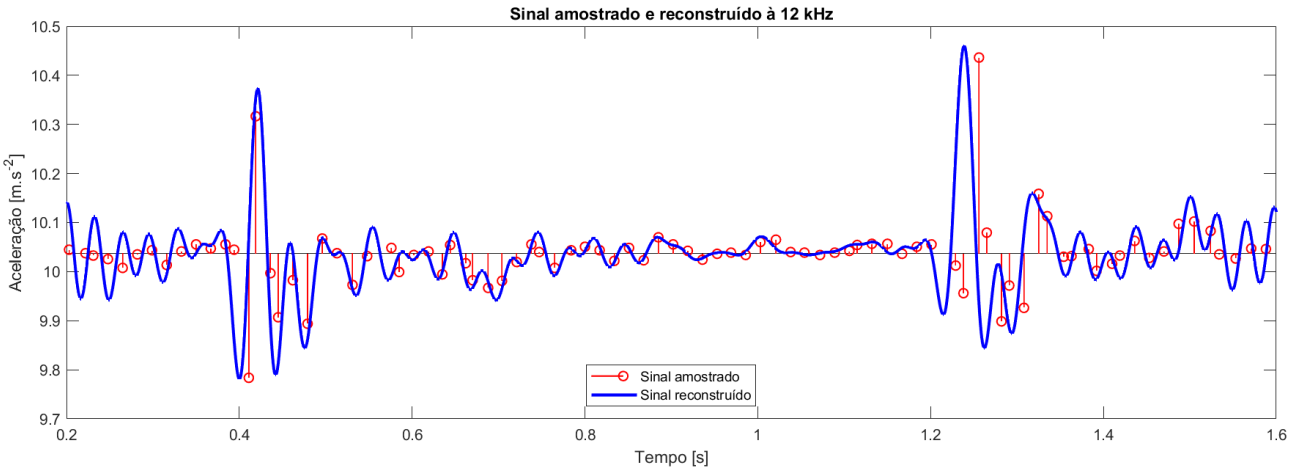

Figura 23: Aceleração captada pelo *smartphone* e sinal reconstruído

Pode-se observar que o ruído do sinal amostrado é muito pequeno em relação ao sinal, mas o ruído do sinal reconstruído é significativamente alto no começo e no fim do sinal. Isto provavelmente se deve ao *alias* das frequências maiores que a metade da frequência de amostragem média. Este comportamento já era esperado, visto que a reconstrução do sinal possui como hipótese que a frequência de aquisição é igual ou superior a frequência de Nyquist não há como garantir esta hipótese com 62 Hz de aquisição.

Além disso, ao reconstruir o sinal em um vetor de tempo com maior resolução, o espaço de armazenamento aumenta proporcionalmente à resolução, sem adicionar mais informações, visto que a reconstrução não irá inserir as ondas de frequência maior que metade da frequência de aquisição.

A partir dessas informações a funcionalidade de reconstrução do sinal em uma frequência maior foi removida da plataforma e ela pode ser implementada dentro do serviço de Loop, caso seja necessário, e não aumentará o espaço de armazenamento.

5 Conclusões

A plataforma desenvolvida foi capaz de coletar os dados de vibração de um *smartphone*, enviá-los a um servidor e retornar a classificação de votla para o *smartphone*, conforme proposto.

Entretanto a frequência média de aquisição foi de 62 Hz, inferior a obtida por Marquardt (2006), que provavelmente se deve ao aplicativo ser desenvolvido na plataforma Expo, que possui mais camadas de abstração para garantir compatibilidade do codigo Android com iOS.

O servidor *cloud* se demonstrou capaz de executar os comandos de classificação de um binário de *Deep Learning* pré-treinado, assim como garantir uma resposta rápida ao aplicativo, tornando improvável que problemas de trafego de dados para a configuração utilizada.

Ao transferir a responsabilidade de treinamento da rede do servidor para uma máquina local, pudemos utilizar uma máquina muito modesta, reduzindo abruptamente o custo deste servidor.

O serviço de *Loop* do servidor é bastante genérico e pode suportar outras aplicações que não sejam necessariamente baseadas em *Deep Learning*, tornando a plataforma mais genérica e útil para outras aplicações.

Referências

[1] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 245–254, ACM, 2006.

[2] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 551–562, ACM, 2011.

[3] I. Google, "Sensors overview | android developers." https://developer.android.com/guide/topics/ sensors/sensors_overview, 2019. Acessado: 2020-01-05.

[4] I. Apple, "Getting raw accelerometer events | apple developer documentation." https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_even 2019. Acessado: 2020-01-05.

[5] I. Amazon, "Instâncias t2 do amazon ec2." https://aws.amazon.com/pt/ec2/instance-types/t2/, 2020. Acessado: 2020-01-05.

[6] K. community, "About keras layers - keras documentation." https://keras.io/layers/about-keras-layers, 2020. Acessado: 2020-01-03.

[7] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.

[8] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in neural information processing systems*, pp. 1096–1104, 2009.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[10] M. Banko and E. Brill, "Scaling to very very large corpora for natural language disambiguation," in *Proceedings of the 39th annual meeting on association for computational linguistics*, pp. 26–33, Association for Computational Linguistics, 2001.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[14] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "Nist cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, pp. 1–28, 2011.