Aplicativo de Tradução de Linguagem de Sinais para Escrita por meio de Processamento de Imagem

Marco Antônio Minotto Ávila Echevestre [1]

[1]Curso de Ciência da Computação – Universidade Franciscana (UFN)
CEP 97010-941 – Santa Maria – RS – Brasil

marcoaminotto@hotmail.com

Abstract. *This work presents the proposal of a system able to perform the translation from images with gestures of the Brazilian Sign Language to writing representation, developed in React Native, Node.js, OpenCV and Brain.js. The system is based on a smartphone application that captures the images with its camera and sends them to a remote server that will perform the image manipulation, obtaining the identification and the respective translation of the gesture, through digital image processing and recognition by artificial neural network. As a result, the system returns the translation of the previously captured gesture to the smartphone.*

Resumo. *Este trabalho apresenta a proposta de um sistema para realizar a tradução de imagens com gestos da Linguagem Brasileira de Sinais (Libras) para a escrita, utilizando React Native, Node.js, OpenCV e Brain.js . Tal sistema é composto por um aplicativo para smartphones que realiza a captação das imagens pela câmera do dispositivo e as envia para um servidor remoto que irá realizar o tratamento das imagens, obtendo a identificação e a respectiva tradução do gesto, por meio do processamento de imagens digitais e de reconhecimento pela rede neural artificial. Como resultado, o sistema retorna, ao smartphone, a tradução para a escrita do gesto previamente capturado.*

1. Introdução

De acordo com pesquisas realizadas pela Organização Mundial da Saúde [Organization 2018], cerca de 446 milhões de pessoas no mundo apresentam alguma forma de deficiência auditiva. Esse número pode ser interpretado como aproximadamente 6,1% da população do planeta. Para esta pequena parte da população, a comunicação é um desafio diferente comparada a outra parte.

Pessoas sem deficiência auditiva, utilizam uma linguagem oral-auditiva, usando a voz, como língua materna para comunicar-se, e pessoas com deficiência auditiva adotam um diferente método de comunicação conhecido como a língua viso-espacial, utilizando as mãos. A linguagem de sinais sendo caracterizada como uma língua viso-espacial, apresenta todos os principais componentes que constituem uma língua oral-auditiva, assim como gramática, semântica, pragmática, sintaxe e entre outras [SILVA et al. 2007].

Com o avanço da tecnologia, atualmente é possível encontrar *softwares* que foram e estão sendo desenvolvidos para auxiliar a interação entre surdo e pessoas sem conhecimento na linguagem de sinais. Bons exemplos disso são os aplicativos de *smartphones*

Handtalk e o *Prodeaf*, que são utilizados tanto por pessoas surdas quanto as não falantes de Libras, com o objetivo de aprender a língua que desconhecem ou que não apresentam um domínio completo[Corrêa et al. 2014].

Nos dias atuais, o desenvolvimento de mecanismos tecnológicos com a função de traduzir as linguagens de sinais chegou a um nível mais complexo. Pesquisadores buscam implementar técnicas avançadas de visão computacional e/ou utilizar componentes eletrônicos de precisão em protótipos para realizar a captação de gestos. Referente a esses estudos, é possível separá-los em duas abordagens distintas para a captação de movimentos, conhecidas por *contact-based* e *vision-based* [Rautaray and Agrawal 2015].

Devido à maior complexidade algorítmica e falta de necessidade em construir um *hardwares* específico para captação de conteúdo, a abordagem *vision-base* vem cativando e desafiando os cientistas a implementarem melhores métodos e estratégias para identificar e reconhecer os gestos linguísticos.

Tendo noção da revisão literária [Neiva and Zanchettin 2018], sendo um estudo de trabalhos acadêmicos que aplicam técnicas de *vision-base* para conhecimento de gestos linguísticos utilizando *smartphones*, demonstra deste modo a possibilidade de desenvolver um aplicativo de *smartphone* para auxiliar a comunicação entre surdos e falantes, sendo que hoje em dia um *smartphone* é um equipamento facilmente encontrado entre as pessoas, de ambas formas linguística, na sociedade.

1.1. Objetivo Geral

Este trabalho tem como objetivo desenvolver um sistema que irá auxiliar a comunicação entre deficientes auditivos, falantes da linguagem de sinais, e de pessoas que não denotam conhecimento nessa língua.

1.2. Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Desenvolver um aplicativo para *smartphone*, capaz de capturar os gestos das mãos;
- Construir um servidor *web* para receber os dados transmitidos pelo aplicativo e processar as informações;
- Implementar técnicas de processamento de imagem no servidor;
- Retornar a tradução escrita do gesto para a tela do *smartphone* do usuário.

2. Referencial Teórico

Esta seção apresenta a revisão bibliográfica sobre o tema escolhido para este projeto, com o objetivo de auxiliar na elaboração da proposta e obter melhor compreensão sobre conceitos e tecnologias que foram utilizadas como base para o estudo deste trabalho.

2.1. Revisão Bibliográfica

Nesta subseção serão abordados conceitos e tecnologias que foram úteis para a construção deste trabalho. Serão apresentados, o conceito de libras, processamento de imagens digitais, redes neurais artificiais, o *framework* React Native para desenvolvimento *mobile*, a tecnologia Noje.js para desenvolvimento *back-end* e a biblioteca OpenCV.

2.1.1. Libras

Assim como as línguas orais-auditivas são distintas dentre países ou até mesmo entre regiões, pode-se afirmar que as linguagens de sinais também apresentam a mesma característica. A partir desse conhecimento mencionado, pode-se apresentar a Língua Brasileira de Sinais (Libras). Definida como a língua materna dos surdos brasileiros pela Federação Nacional de Educação e Integração de Surdos (FENEIS) [SILVA et al. 2007].

A Libras surgiu após um professor surdo francês e praticante da *Língua de sinais Francesa* (LSF), conhecido por *Professor Ernest Huet*, ter chegado ao Brasil em 1856. Após concluir que o país necessitava de uma língua de sinais própria, começou a trabalhar nessa idéia e assim desenvolveu, tendo como base a LSF, a Libras[Monteiro 2006].

A estrutura da Libras é formada por parâmetros primários e secundários que funcionam em conjunto de forma sequencial ou simultânea. Os parâmetros primários são classificados como as configurações das mãos, o ponto de articulação e movimento. Já os parâmetros secundários têm a disposição das mãos, orientação da palma das mãos, região de contato e expressões faciais [Brito 1995].

2.1.1.1 Processamento de Imagens Digitais

O Processamento de Imagens Digitais [Petrou and Petrou 2010] é uma área da computação gráfica que é utilizada para manipular imagens e *frames* de vídeos por meio de algoritmos e assim gerar uma nova representação das mesmas.

Os algoritmos utilizados podem ser entendidos como técnicas, que apresentam finalidades distinta. Essas técnicas podem ser utilizadas para realizar transformações geométricas, manipular cores, contraste, luminosidade, realizar interpolação, aplicar métodos de segmentação para extrair dados que possam ser úteis para realizar identificação de objetos e entre outras finalidades.

Ao implementar algoritmos de processamento de imagens digitais é possível solucionar problemas em diversas áreas, tais como na saúde, mas precisamente na imagiologia e biometria, na astronomia, principalmente implementada nos satélites, na segurança, encontrado em sistemas de vigilância na qual podem realizar a detecção de faces, nos videogames, utilizado no aparelho *kinect* do console XBOX para identificar a pessoa e seus movimentos, e assim por diante.

2.1.2. Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) [Haykin 1994] são um ramo da área de Inteligência Artificial, na qual trabalha com modelos matemáticos inspirados nas estruturas neurais biológicas e apresentam capacidade computacional adquirida por meio de aprendizado e generalização.

Essa área de conhecimento nasceu nas décadas de 40 e 50, por meio da publicação de três artigos realizados por diferentes pesquisadores, na qual apresentaram o primeiro modelo de redes neurais simulando máquinas, o modelo básico de rede de auto-

organização e o modelo Perceptron de aprendizado supervisionado.

O funcionamento de uma rede neural ocorre baseado nas interações que ela apresenta com as informações externas. O processo de aprendizado de uma RNA é realizado de modo iterativo e a partir desse modo a rede neural deve gradativamente aprimorar-se à medida que ela interpreta cada informação externa recebida. Para diagnosticar se o processo de aprendizado será realizado adequadamente é apresentado o critério de desempenho, por ele é determinado o ponto de parada de treinamento e a qualidade do modelo neural na qual são preestabelecidos pelos parâmetros de treinamento. E por meio desse treinamento repetitivo é esperado que a RNA trabalhada seja capaz de reconhecer os dados externos fornecidos.

2.1.3. React Native

React Native [Source 2019] é um *framework* utilizado para o desenvolvimento de aplicações nativas em sistemas Android e iOS. Este *framework* foi criado com base em uma biblioteca JavaScript *open source* conhecida como React, na qual é usada para desenvolver interfaces de usuário no *browser*. Ambas elas foram criadas e utilizadas desde então pelo Facebook.

Semelhante ao desenvolvimento para React para a *web*, as aplicações em React Native são escritas em JavaScript e em uma mistura desta linguagem de programação com *Extensible Markup Language* (XML) Schema, conhecida como JavaScript XML (JSX). Com a utilização desta tecnologia, os desenvolvedores podem criar *views* nativas e acessar componentes específicos da plataforma nativa por meio de JavaScript.

Atualmente é uma ferramenta que vem sendo utilizada por milhares de aplicações disponíveis *online*, além de *apps* de grandes empresas, tais como: Tesla, Walmart, Microsoft, Bloomberg e entre outras[Dabit 2019].

2.1.4. Node.js

Node.js [Foundation 2019] é um ambiente de execução JavaScript construído em uma *engine* de JavaScript e WebAssly de alta performance e *open source* conhecida por V8, pertencente ao Google. Com a sua criação, desenvolvida pelo Ryan Dahl em 2009, permitiu com que o JavaScript pudesse ser utilizada para a área de desenvolvimento em back-end em aplicações web, já que anteriormente essa linguagem era utilizada somente para o front-end.

Essa tecnologia possui uma arquitetura orientada a eventos capaz de controlar operações de entrada e saída assíncronas, criada com o propósito de permitir o desenvolvimento de aplicações escaláveis de rede. Com essa abordagem, o Node é uma ferramenta para implementar aplicações *web real-time*, já que ele consegue otimizar a vazão e escala de requisições em aplicações *web* que apresentam uma grande quantidade de operações de entrada e saída.

2.1.5. OpenCV

OpenCV [team 2019] é uma biblioteca *open-source* escrita em C++ utilizada para visão computacional e aprendizado de máquina. A sua criação teve como objetivo de fornecer uma infraestrutura comum para aplicações que utilizam visão computacional e para ampliar o uso de *machine perception* em produtos comerciais.

Essa biblioteca contém mais de 2500 algoritmos de otimização, os quais podem ser utilizados para detectar e reconhecer faces, identificar objetos, classificar ações humanas por vídeo, rastreamento de movimento em câmera, rastreamento de movimento de objetos, extração de modelos três dimensões (3D) de objetos e entre outras diversas funções possíveis.

2.2. Trabalhos Correlatos

Esta subseção apresenta os trabalhos correlatos que são utilizados como base para o desenvolvimento deste trabalho. Os trabalhos, em questão, utilizam diversas abordagens para realizar a captação, identificação e tradução dos gestos da linguagem de sinais por meio de técnicas de processamento de imagem.

2.2.1. A Mobile Application of American Sign Language Translation via Image Processing Algorithms

O projeto [Jin et al. 2016] foi proposto em cooperação entre a Universiti Teknologi Malaysia e a Monash University Malaysia, numa tentativa de desenvolvolver um aplicativo para *smartphones* com o objetivo de identificar e traduzir dezesseis gestos do alfabeto da *American Sign Language*(ASL), por meios de técnicas de visão computacional. O projeto consiste de um *smartphone* Nokia Lumia 1520 com o sistema operacional Windows Phone 8.1, que foi utilizado para captar imagens e simplificá-las a uma resolução de 320 x 240 pixels no formato *Red, Green and Blue* (RGB). O algoritmo foi desenvolvido baseado na biblioteca EmguCV, que trata-se de um sistema multiplataforma .NET com a integração do processamento de imagem da biblioteca OpenCV.

Para realizar a identificação dos gestos, são utilizados alguns métodos de processamento de imagem denominados de calibração, segmentação, extração de características e classificação. Como resultado, conseguiram entregar um *framework* que consegue identificar com uma precisão média de 97,13% dezesseis letras do alfabeto da ASL, e provar, baseado nos experimentos da pesquisa, que o método de extração de características SURF é mais eficiente e preciso computacionalmente que o método SIFT.

2.2.2. Selfie Video Based Continuous Indian Sign Language Recognition System

O projeto [Rao and Kishore 2018] proposto pelo *Department of Electronics and Communications Engineering* pertencente a K.L. University. O trabalho desenvolvido teve como proposta a criação de um aplicativo para *smartphone* com o objetivo de identificar os gestos da linguagem de sinais do usuário e traduzi-lo, pela utilização de técnicas em visão computacional aplicada a vídeo.

O algoritmo de reconhecimento utilizado baseia-se em cinco parâmetros, e eles são: o reconhecimento da mão e da cabeça; a orientação da mão e da cabeça; movimento da mão; forma da mão e localização da mão e da cabeça. Com a utilização desses parâmetros, o projeto pode identificar os gestos com uma melhor precisão, classificá-los e fornecê-los para uma rede neural artificial para serem treinados.

Para a gravação de vídeo, foi utilizada a câmera frontal dos smartphones *Asus Zen phone II* e o *Sansung Galaxy S4*, posicionados em um *selfie stick*, ambos equipados com 5MP nas câmeras frontais. Para realizar a captura de vídeo, foi realizada em uma sala, com uma iluminação ambiente e cenário de fundo unicolor. Foi também utilizado para executar o *software*, em curto periodo de tempo, num computador de 4GB RAM. Foi usado um banco de dados formal com 18 gestos da linguagem de sinais representadas continuamente, capturadas por 10 diferentes falantes desta linguagem.

Como resultado o projeto registrou em torno de 85.58%, por intermédio da distância mínima de classificação, e 90.58%, por meio da rede neural artificial, de precisão em combinação de palavras utilizando o método de Distância de Mahalanobis em menos de 0.6s, na qual apresentou um melhor resultado comparado com os métodos de Distância Euclidiana e Distância Euclidiana Normalizada.

2.2.3. Real Time Hand Gesture Recognition Using Different Algorithms Based on American Sign Language

O projeto [Islam et al. 2017] proposto pelo *Department of Electrical and Eletronic Engineering* pertencente a *Shahjalal University of Science and Technology*, tendo como proposta desenvolver um sistema de reconhecimento de sinais da ASL em tempo de execução cujo a ênfase foi a utilização de um novo algoritmo para identificar gestos. Esse novo algoritmo proposto é responsável por realizar a localização das pontas dos dedos e foi denominado de "*K convex hull*", graças ao resultado da junção dos algoritmos de *K Curvature* e de *Convex Hull*.

Houve, também, a implementação de outros algoritmos para realizar a extração de características, tais como: rotação, excentricidade, *elongatedness measurement* e segmentação de *pixels*. Além disso, nesse projeto foi utilizada uma *Artificial Neural Network* (ANN), alimentada com 1850 imagens feitas por 37 pessoas distintas, com a função de realizar o reconhecimento dos gestos em tempo de execução. Para a sua construção, foi aplicado o algoritmo *back propragation* para realizar o treinamento, e utilizado como fonte o banco de dados populado com características de diversos sinais da ASL.

Para realizar a captação de novos sinais desconhecidos, articulados pelo usuário, é utilizado um aplicativo Android chamada DroidCam, que permite usar a câmera do celular para captar as imagens em tempo de execução, com uma melhor qualidade, dos gestos captados. Um critério importante para esse processo nesse trabalho é fazer a captação com um fundo escuro como cenário. Para avaliar os novos sinais captados, interpretá-los e para demonstrar os resultados obtidos visualmente, foi desenvolvido uma *graphical user interface(GUI)*.

A ANN utilizada foi implementada com diferentes algoritmos para realizar testes e

compará-los. Os algoritmos foram o *K convex hull*, o *Eccentric & Elongatedness*, o *Pixel segmentation* e por fim a combinação de todos eles. Como resultado que teve o melhor desempenho foi a junção de todos ele, com precisão aproximada de 99.5% em teste, validação e reconhecimento médio. Em relação a precisão do reconhecimento médio em tempo de execução foi de 94.32%.

2.2.4. Considerações sobre os trabalhos relacionados

Os trabalhos mencionados nesta seção apresentam um mesmo propósito final, que é realizar a tradução de gesto da linguagem de sinais com a melhor precisão de acerto possível utilizando um *smartphone* como equipamento de captação. Esses trabalhos seguem um número de etapas necessárias para alcançar o objetivo final, podendo elas serem classificadas em captação, segmentação, extração de detalhes e classificação.

Baseado nessas etapas, os trabalhos se distinguem ao aplicar diferentes formas para concluí-las. Enquanto alguns projetos utilizam como modo de captação uma imagem para ser processada, como no trabalho do [Jin et al. 2016], outros utilizam um vídeo, apresentado no trabalho do [Rao and Kishore 2018]. Ao realizar uma escolha inicial como essa, esses projetos tiveram que aplicar diferentes abordagens nas distintas etapas para conseguirem apresentar um resultado final. Como mencionado na revisão literária [Neiva and Zanchettin 2018], *"os trabalhos que abordam a identificação de gestos estáticos e gestos dinâmicos, não utilizam os mesmos métodos de reconhecimento"*. Com isso optar pela captação por vídeo pode exigir maior processamento computacional para realizar o reconhecimento do gesto, pelo motivo de uma foto ser somente um *frame* a ser processados pelas etapas mencionadas, enquanto um vídeo pode ser separado em diversos *frames* com o mesmo fim da foto.

Apesar das distintas abordagens presente em cada um desses projetos, foi possível identificar um fator semelhante entre eles, que trata-se da escolha de realizar a captação das imagens em um cenário de fundo simples (unicolor), o que acaba sendo uma vantagem, pois não necessitam utilizar tantas técnicas de processamento de imagem para distinguir a mão do usuário com o cenário. Tendo como conhecimento essa abordagem, a precisão de seus algoritmos obtiveram uma média de reconhecimento dos gestos acima de 80%, o que talvez pudesse ser distinto ao aplicar os algoritmos com um cenário multicolor.

3. Proposta

Com o avanço tecnológico, as possibilidades de facilitar a vida de quem apresenta dificuldade para compreender as linguagens de sinais são maiores comparado com as décadas anteriores, onde somente descobriria sobre assunto se interagisse com algum livro educativo sobre a linguagem ou entrasse em contato com alguém que pudesse ensina-lá.

Dessa forma, este trabalho propõe o desenvolvimento de um sistema com o objetivo de traduzir imagens que contenham gestos pertencentes a língua brasileira de sinais para a escrita visual. A Figura 1 ilustra uma visão geral da proposta.

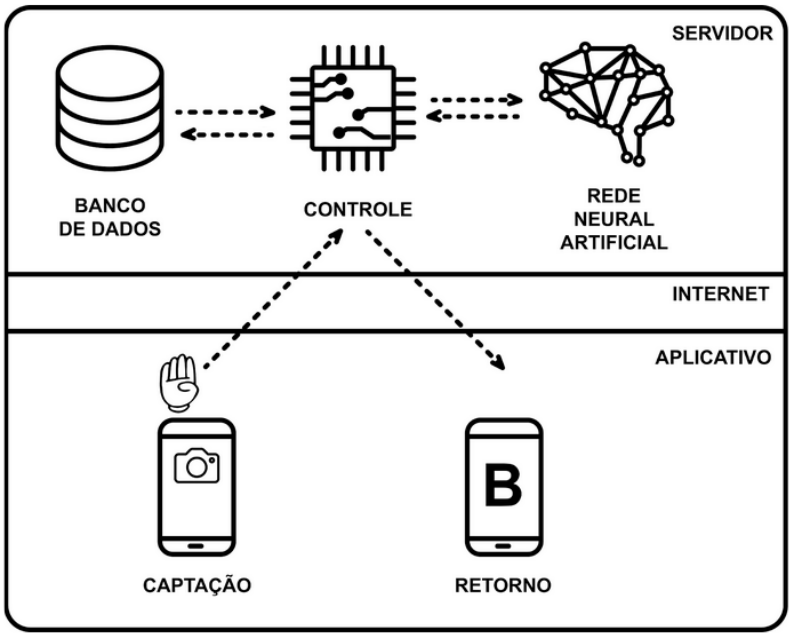

Figura 1. Visão geral da Proposta

As etapas apresentadas na Figura 1 representam a sequência de eventos que ocorrerá no projeto. O processo inicia com a captação, onde o usuário do aplicativo salva imagens do deficiente auditivo, gesticulando um determinado sinal, por meio da utilização da câmera do *smartphone*. Ao realizar este processo, o aplicativo envia a foto pela Internet para o servidor *web RESTful*. Esse servidor está conectado a um banco de dados populado com diversas imagens de sinais, na qual é usada para treinar a rede neural artificial implementada.

No momento que o servidor recebe a imagem, são aplicadas diversas etapas de processamento de imagens para extrair características da mesma. Ao gerar esses dados, o servidor os passará para a rede neural artificial para que possa realizar a classificação. Ao ser identificado o gesto, a RNA retorna o resultado para o servidor, que posteriormente enviará a resposta ao aplicativo para que assim possa mostrar ao usuário.

Além disso o sistema apresenta a proposta de ter um aplicativo que não exige elevado processamento do *smartphone*, pelo motivo de não ter de executar o processamento de imagem e a RNA nele, já que isso é feito pelo servidor. Assim resultando em um aplicativo que possa ser instalado em smartphones não tão modernos, necessitando ter pelo menos no seu hardware uma câmera traseira e utilizar o sistema operacional Android ou IOS, devido o desenvolvimento por meio do React Native.

4. Metodologia

A metodologia escolhida para auxiliar no desenvolvimento deste projeto é uma adaptação da metodologia *Scrum* [Pries and Quigley 2010], na qual trata-se de um *framework* utilizado para o planejamento e gestão de projetos de *software* desde o início de 1990. No *Scrum* os projetos que o aplicam, são divididos em ciclos, denominados de *Sprints*. E cada Sprint apresenta um conjunto de atividades e um tempo limite para serem finalizadas. Assim considerada uma metodologia ágil, tem como objetivo de melhorar a produtividade e agilidade da equipe para entregar um produto qualificado no final.

A adaptação mencionada é o resultado da mistura das metodologias *Scrum* e da *Personal Software Process* (PSP), na qual foi denominada de *Scrum Solo* [Pagotto et al. 2016]. Esse projeto foi proposto pela Universidade Tecnológica Federal do Paraná, com o objetivo de suprir a ausência de uma metodologia para o auxílio no desenvolvimento de *software* realizado por apenas um programador.

Ao visualizar a Figura 17 localizada no Anexo I, é possível perceber a semelhança em ambas metodologias. Tanto que as etapas do *product backlog* e do *sprint backlog* funcionam de maneira semelhante entre esta metodologia e a *Scrum*. Outro exemplo de semelhança está no final dos *sprint*, em que o desenvolvedor deve entregar um protótipo do *software* com novas funcionalidades ao usuário.

Mas a diferença é visível ao ser aplicado a redução de tempo referente a execução dos *sprints*, na qual o tempo não deve passar de uma semana para serem finalizados. Outro fator distinto de importante relevância é a inexistência de reuniões diárias, devido ao desenvolvimento solo.

O planejamento deste projeto, por meio da metodologia *Scrum Solo*, é descrito em três etapas conhecidas como Requisitos, *Sprint* e Implementação. As duas primeiras etapas podem ser encontradas, na ordem mencionada, nos apêndices I e II, já a Implementação é descrita na subseção a seguir.

4.1. Implementação

Nesta subseção é fornecida as informações de como foi implementado neste projeto, assim detalhando separadamente em subseções cada camada do sistema.

4.1.1. Cliente

O aplicativo[1] desenvolvido em React Native apresenta uma interface na qual permite ao usuário capturar uma foto e enviá-la para o servidor, ou capturar uma nova, assim representada na Figura 2. Com isso o próprio usuário consegue analisar se a foto capturada estava em boas condições para ser diagnosticada pelo sistema, assim evitando que imagens tremidas, fora de foco ou sem a presença da mão sejam processadas pelo servidor.

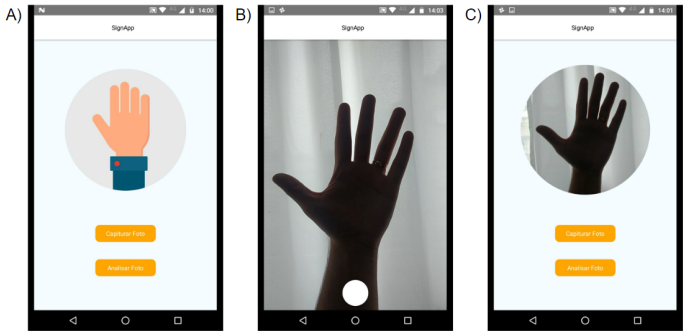

Figura 2. Telas do aplicativo

[1]Código da implementação do aplicativo está disponível no GIT: https://github.com/marcoaminotto/signappcamera

Para realizar a captação da imagem, foi desenvolvida a função assíncrona *takePicture*, encontrada no código da Figura 3, na qual aciona a câmera traseira do *smartphone* quando o usuário clica no botão da tela B da Figura 2. Ao verificar se a câmera foi acionada, é realizada a configuração referente a qualidade e o armazenamento da imagem, na qual é utilizada o formato *base64*. A partir desse ponto, é acionada a função *takePictureAsync* da biblioteca React Native Camera [Wansbrough 2019] para capturar a imagem, passando como parâmetro as configurações registradas. Com isso é salvo no *state* da classe a imagem registrada e altera o valor da variável *show* para *false*, assim voltando para a tela C da Figura 2, e já substituindo o desenho da mão para a imagem capturada.

```
// Função que realiza a captura da imagem
takePicture = async() => {
  if (this.camera) {
    const options = { quality: 0.5, base64: true };
    const img = await this.camera.takePictureAsync(options);
    console.log(img.uri);
    this.setState({ img, show: false });
  }
};
```

Figura 3. Função que realiza a captação da imagem

Após a imagem da mão ter sido capturada, a mesma é enviada para ser processada no servidor. Para isso, o usuário terá de clicar no botão "Analisar Foto" da tela C da Figura 2, que assim acionará a função *upload*, cujo trecho de código está apresentado na Figura 4.

Na função *upload* é criada um *FormData* e atribuido à ela a imagem com o nome modificado para a data e hora do *smartphone*, e o sinal com valor nulo. Por meio da *API Fetch*, do *Javascript* é enviado uma requisição do método *POST* para o servidor, passando o *FormData* como formato *multipart*, pelo motivo de conter uma imagem. Após o envio, é informado um *Alert* sobre o resposta do método na tela do *smartphone*.

```
// Função que envia os dados para o servidor
upload = () => {
  const data = new FormData();
   //pegar a imagem da store e passar para o state
   let name = new Date().getTime();
   data.append('image', {uri: this.state.img.uri, name: `${name}.png`, type: 'image/png' });
   data.append('sign',null);
   fetch('http://192.168.25.8:1111/posts', {
    method: 'post',
    headers: {
      'Content-Type': 'multipart/form-data'
    },
    body: data
   }).then(a => a.json()).then(res => alert(res));
}
```

Figura 4. Função que realiza o envio da imagem ao servidor

4.1.2. Servidor

O servidor[2] foi desenvolvido em Node.js foi baseado no padrão de arquitetura de *software* denominado de *Model-View-Controller* (MVC), assim apresentando a característica do sistema está dividido em três camadas de distintas responsabilidades.

A camada *Model* está relacionada a manipulação de dados, assim responsável por ler e escrever informações no banco de dados. Referente a esse tópico é importante mencionar que esse projeto está utilizadando o MongoDB Atlas [Atlas 2019], que é um banco de dados *NoSLQ* hospedado em nuvem. E para realizar a comunicação entre o servidor e o banco de dados MongoDB, foi implementada a biblioteca Mongoose, que é uma ferramenta para modelar objetos do MongoDB desenvolvida para trabalhar de forma assíncrona. Por meio dessa biblioteca, o sistema consegue estabelecer a conexão com o MongoDB, além de montar a estrutura do banco e definir os tipos de dados de entradas, assim demonstrado no código da Figura 5.

```
3  const PostSchema = new mongoose.Schema({
4      image: String,
5      sign: String,
6      binarry: Buffer
7  },{
8      timestamps: true,
9  });
10
```

Figura 5. Estrutura de dados que são armazenados no banco de dados

A camada *View* é responsável pela interação com o usuário, assim tendo a função de exibição de dados. Mas como esse projeto está utilizando o aplicativo como meio de comunicação entre usuário e sistema, então no servidor não foi implementado nenhuma *view*, assim fazendo com que o servidor não necessite apresentar implementação para realizar o *frontend*.

A camada *Controller* é responsável de receber as requisições do usuário, assim tendo a responsabilidade de controlar qual *model* utilizar e qual *view* mostrar ao usuário. Nesse projeto, o *Controller* realiza o controle do *model*, manipulação dos dados enviados pelo aplicativo e retorno de informações ao aplicativo. Por meio de uma função POST, o *controller* recebe do aplicativo uma foto na qual ela passa pela etapa de processamento de imagem e, assim quando termina, é salva localmente e os dados obtidos dela são salvos no banco de dados. Há também uma função GET responsável por buscar todos os dados salvos do banco de dados em um JSON e enviá-los para onde foi chamada a função.

4.1.2.1 Processamento de imagem

O processamento de imagem é realizado quando o aplicativo envia a foto para o servidor, assim enviando a foto e mais outras informações, tais como o sinal com valor nulo e o nome da imagem, juntas em um formato JSON. O servidor ao receber o arquivo, realiza

[2]Código da implementação do servidor está disponível no GIT: https://github.com/marcoaminotto/webserver

a extração da imagem do arquivo e inicializa o processo de tratamento da imagem. Esse processo é possível ser visto na Figura 6.

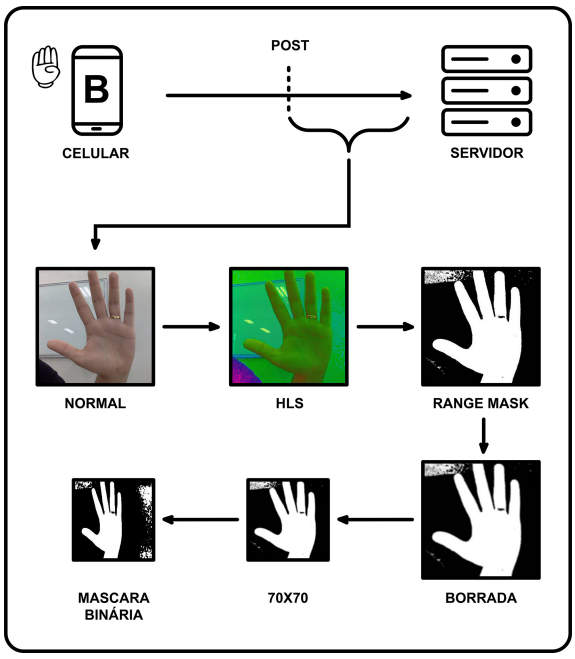

Figura 6. Etapas do processamento de imagem

A primeira etapa consiste em transformar a imagem recebida no formato RGB para uma versão alternativa conhecida por *Hue, Lightness, Saturation* (HLS). Com ela é criada a primeira versão da máscara binária, por meio da identificação da variedade de cores que se aproximam do tom da pele humana. Assim resultando em uma imagem preto e branca, na qual os *pixels* que se assemelham a pigmentação da pele, são transformados em branco e os demais em preto. A seguir é retirado o ruído da máscara binária, aparentando ter a característica de estar mais borrada, e portanto, tornando o contorno da mão mais suave. A seguir, a imagem sem ruído é redimensionada para uma nova máscara binária de 70x70 pixels. E por fim é gerada uma nova máscara binária pelo método de *threshold* que é utilizada para extrair os dados.

Para identificar a mão, foi testada a variação de cores em HLS que mais se aproximariam da cor da pele humana. Apesar disso não é uma identificação perfeita, já que ao mudar o cenário de captação é necessário alterar os dados de luminosidade e saturação, localizado nas funções *skinColorUpper* e *skinColorLower* demonstrado no código da Figura 7.

```
const skinColorUpper = hue => new cv.Vec(hue, 0.8 * 255, 0.6 * 255);
const skinColorLower = hue => new cv.Vec(hue, 0.1 * 255, 0.05 * 255);

const mat = cv.imread(req.file.path);
const imgHLS = mat.cvtColor(cv.COLOR_BGR2HLS);
//cv.imwrite(req.file.path, imgHLS);
 const rangeMask = imgHLS.inRange(skinColorLower(0), skinColorUpper(24));
```

Figura 7. Etapas do processamento de imagem

A decisão de redimensionar a foto após já obter a imagem sem ruidos foi tomada após testar o redimensionamento em todas as etapas, e a que obteve a melhor nitidez da mão foi ao aplicar o redimensionamento da imagem depois de obter a foto borrada, assim apresentada na Figura 8.

Figura 8. Primeira imagem com redimensionamento após etapa de retirada de ruídos e a segunda imagem com redimensionamento ao receber a foto do aplicativo

Com a máscara binária gerada, começa o processo de extração de dados. Logo após o término do processamento de imagem, o algoritmo irá obter a imagem 70x70 e irá extrair o valor dos *pixels* em uma matriz de 70 por 70. Os dados salvos nela são compostos dos números 0 ou 255, assim representando as cores branco e preto. Na próxima etapa a matriz é transformada em um vetor de 4900 posições, e é aplicada a técnica de normalização dos dados. Nesse caso, os valores dos dados serão substituídos de 255 para 1, resultando na redução de espaço em memória e facilitando o treinamento e identificação dos dados pela RNA.

4.1.2.2 Treinamento

O treinamento da RNA acontece quando é inciado o servidor, na qual realiza a busca das máscaras binarios e seus respectivos sinais cadastrados do banco de dados e às utiliza como dados de entrada para treinar a rede neural. O tipo de RNA implementada neste projeto foi uma rede neural *feedforward* com *backpropagation* processada pela GPU, por meio da biblioteca Brain.js[Plummer 2019]. Essa rede neural foi estruturada no formato *multilayers*, sendo assim organizada em: uma camada de entrada, três camadas intermediárias e uma de saída.

Para alimentar a rede neural, foi desenvolvido um *dataset* com 500 imagens captadas pelo próprio aplicativo desenvolvido neste projeto. A partir dessas imagens foram geradas 500 máscaras binárias, cada uma contendo 4900 elementos, que são utilizados como dados de entrada para a camada de entrada. Para as camadas intermediárias, foi designado 3267 neurônios para cada uma das três, para que assim consiga obter uma base de conhecimento consistente. E por fim apresentando 10 resultados na camada de saída, representando cada letra das que foram escolhidas para fazer o reconhecimento.

O treinamento da RNA é um aprendizado supervisionado, na qual é informado o resultado desejado de cada dado de entrada. Para isso a função de ativação utilizada é a Sigmoid, assim permitindo obter uma precisão do aprendizado com base nos dados de entrada e o tempo de interações entre eles, e como resultado gerando o conhecimento

na rede neural. Esse processo de treinamento acontece ao iniciar o servidor, na qual o sistema busca as máscaras binárias salvas no banco de dados e as utiliza para gerar o conhecimento da rede neural. Para realizar o treinamento, o servidor está hospedado em um computador com uma placa gráfica GeForce GTX 960 Windforce com 4GB GDDR5 dedicada e um processador Intel Core i5, e levando em torno de 5 segundos para realizar 10 interações.

5. Cenário de Validação e Testes

Nessa seção são citados os requisitos para validar se o produto final deste trabalho será um *software* que cumpre com os objetivos do projeto. E portanto para verificar isso, a validação será baseada na identificação da mão ao realizar o processamento de imagem e nos acertos obtidos ao identificar os gestos recebidos pelo aplicativos.

O teste de identificação da mão, foi realizado em um fundo branco, com uma distância máxima de 60cm e mínima de 25cm do *smartphone*. Além disso o teste foi realizado a noite, aproximadamente às 19h, e por tanto a iluminação foi feita por um abajour com uma lâmpada de 15W. Assim visto na imagem na Figura 13 do Apêndice III.

Para validar a RNA, o teste ideal seria o usuário que iria testar o aplicativo do *software*, realizar a gesticulação das letras A, B, C, D, E, F, G, I, L, V do alfabeto da Língua Brasileira de Sinais, assim escolhidas por serem representadas por gestos estáticos. Seriam capturadas, pelo *smartphone*, 10 imagens de cada gesto das letras citadas anteriormente, e seria esperada uma resposta com o resultado correto de cada. Se a precisão média de acertos dos sinais fosse entre 90% a 100%, seria considerado um *software* com uma precisão de classificação alta. Essa avaliação é baseada no estudo da revisão literária de [Neiva and Zanchettin 2018], cujo trabalho realizou um cálculo de média com as precisões de acertos obtidas na fase de classificação dos 14 projetos de reconhecimento de gestos estáticos, encontrados no trabalho citado. E por fim o resultado foi de aproximadamente 90%.

Mas como a RNA implementada nesse sistema não obteve um bom resultado, os testes foram realizados apenas com a letra ”a”, utilizando o mesmo cenário para validar o processamento de imagem do sistema. A explicação das dificuldades encontradas para tentar obter um bom resultado pode ser encontrada na subseção Problemas e Dificuldades.

6. Conclusão

Este projeto apresentou a proposta de um aplicativo para realizar a tradução de gestos da linguagem de sinais para a escrita, abordando assuntos como processamento de imagens digitais e rede neurais artificiais.

Foram realizadas pesquisas sobre o desenvolvimento de aplicações que visavam a tradução da linguagem de sinais por meio de distintos métodos de captação e processamento de imagens. Os trabalhos estudados abordavam tanto reconhecimento de gestos estáticos, realizados por imagem, quanto de gestos dinâmicos, realizados por vídeo.

Dentro do contexto proposto nesta abordagem, é possível destacar algumas vantagens, tendo como principal disponibilizar para a comunidade um aplicativo que consegue identificar a mão humana e extrair dados a partir dela, além de apresentar também técnicas para realizar o reconhecimento de gestos da LIBRAS, sem sobrecarregar o

smartphone com muito processamento pois não realiza o processamento de algoritmos de identificação de imagens e classificação, já que o servidor está responsável por esta função. E com isso o aplicativo pode funcionar em *smartphone* de diversas gerações, desde que o aparelho tenha uma câmera e acesso à internet.

Esse projeto conseguiu cumprir com 3 dos 4 objetivos específicos estabelecido. Apenas não concluindo o último deles, referente a retornar ao aplicativo a tradução para a escrita do gesto identificado, devido a complexidade do treinamento da RNA. Para obter um treinamento preciso seria necessário mais recurso de *hardware* e/ou realizar mais pesquisas para descobrir outras abordagens para realizar um treinamento de rede neural que retorne um resultado satisfatório. Na subseção Problemas e dificuldades é possível encontrar uma descrição detalhada do problema relacionado ao treinamento.

Em relação a trabalhos futuros há algumas sugestões, tais como: melhorar o treinamento da RNA por meio do aumento na quantidade de fotos no *dataset* ou extrair mais parâmetros da imagem e usá-los para treinar a rede neural, além da máscara binária; Exemplos de dados que possam ser considerado para melhorar o treinamento é utilizar o contorno da mão, quantidades de pontos e distância entre os pontos mais altos e mais baixos como dado de entrada para a RNA, função que já está implementada no código do servidor deste projeto. O resultado dessa função pode ser vista na Figura 16 localizada no Apêndice VI; Uma outra sugestão seria utilizar alguns usuários com deficiência auditiva para testar o aplicativo, e verificar se realmente é útil para melhorar a comunicação. Assim pode ser feito um levantamento de requisitos para melhorar o sistema.

6.1. Problemas e Dificuldades

No início do projeto, era planejado utilizar máscaras binárias a partir de imagens com dimensionamento de 260x260, na qual equivale a um vetor de 67600 posições. Mas a RNA implementada neste projeto não conseguiu iniciar o processo de interações, devido a grande quantidade de dados. Com isso foram realizados testes para conseguir identificar a resolução que levaria um tempo aceitável para realizar o treinamento e ainda mantivesse a precisão ao gerar a máscara binária da mão. Os resultados dos testes verificados estão apresentados na Figura 14 localizada no Apêndice IV. E a partir desses testes foi decidido adotar a resolução 70x70.

Um problema que ocorreu no final do projeto e não solucionado, foi identificar a quantidade de camadas intermediárias e o número de neurônios necessários em cada uma delas para realizar um treinamento aceitável que conseguisse identificar a tradução do gesto fornecido. Com isso foram realizados diversos testes utilizando a função de ativação Sigmoid, para construir um treinamento com base em 50 máscara binárias de cada sinal, e após 500 interações foi realizado o teste para identificar a letra ”a”, e os resultados obtidos são apresentados na Figura 15 do Apêndice V.

Referências

Atlas, M. (2019). Mongodb atlas. Disponível em: `https://www.mongodb.com/cloud/atlas`. Acesso em: 30 jul. 2019.

Brito, L. F. (1995). *Por uma gramática de línguas de sinais*. Tempo Brasileiro.

Corrêa, Y., Vieira, M. C., Santarosa, L. M. C., and Biasuz, M. C. V. (2014). Aplicativos de tradução para libras e a busca pela validade social da tecnologia assistiva. In *Bra-*

zilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE), volume 25, page 164.

Dabit, N. (2019). *React Native in Action*. Developing iOS and Android apps with JavaScript. MANNING Shelter Island, NY, USA.

Foundation, N. (2019). About node.js. Disponível em: `https://nodejs.org/en/about/`. Acesso em: 30 abr. 2019.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Islam, M. M., Siddiqua, S., and Afnan, J. (2017). Real time hand gesture recognition using different algorithms based on american sign language. In *2017 IEEE international conference on imaging, vision & pattern recognition (icIVPR)*, pages 1–6. IEEE.

Jin, C. M., Omar, Z., and Jaward, M. H. (2016). A mobile application of american sign language translation via image processing algorithms. In *2016 IEEE Region 10 Symposium (TENSYMP)*, pages 104–109. IEEE.

Monteiro, M. S. (2006). História dos movimentos dos surdos e o reconhecimento da libras no brasil. *ETD-Educação Temática Digital*, 7(2):292–305.

Neiva, D. H. and Zanchettin, C. (2018). Gesture recognition: a review focusing on sign language in a mobile context. *Expert Systems with Applications*, 103:159–183.

Organization, W. H. (2018). Deafness prevention. Disponível em: `https://www.who.int/deafness/estimates/en/`. Acesso em: 21 mar. 2019.

Pagotto, T., Fabri, J. A., Lerario, A., and Gonçalves, J. A. (2016). Scrum solo: software process for individual development. In *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE.

Petrou, M. M. and Petrou, C. (2010). *Image processing: the fundamentals*. John Wiley & Sons.

Plummer, R. (2019). Brain.js: Gpu accelerated neural networks in javascript. Disponível em: `https://github.com/BrainJS/brain.js`. Acesso em: 15 out. 2019.

Pries, K. H. and Quigley, J. M. (2010). *Scrum project management*. CRC press.

Rao, G. A. and Kishore, P. (2018). Selfie video based continuous indian sign language recognition system. *Ain Shams Engineering Journal*, 9(4):1929–1939.

Rautaray, S. S. and Agrawal, A. (2015). Vision based hand gesture recognition for human computer interaction: a survey. *Artificial intelligence review*, 43(1):1–54.

SILVA, F. I. d., REIS, F., Gauto, P., Silva, S. d. L., and Paterno, U. (2007). Aprendendo língua brasileira de sinais como segunda língua. *Santa Catarina: NEPES*.

Source, F. O. (2019). React native. Disponível em: `https://facebook.github.io/react-native/`. Acesso em: 23 abr. 2019.

team, O. (2019). About opencv. Disponível em: `https://opencv.org/about/`. Acesso em: 30 abr. 2019.

Wansbrough, L. (2019). React native camera. Disponível em: `https://github.com/react-native-community/react-native-camera`. Acesso em: 12 jun. 2019.

Apêndice I

Nesta fase do projeto é realizado um estudo no planejamento de implementação do *software* proposto, que são representados pelos artefatos escopo, *product backlog* e protótipo do *software*.

A. Escopo

Apesar de haver uma grande quantidade de pesquisas relacionadas ao desenvolvimento de aplicativos para auxílio na comunicação com deficientes auditivos, ainda não há um produto no mercado que tenha se destacado por identificar gestos por meio de imagens. Com esta finalidade, o *software* proposto por este trabalho apresenta uma solução para traduzir gestos de uma pessoa pelas fotos capturadas pelo aplicativo. Assim o *software* pode apresentar as finalidades de auxiliar em meio a uma comunicação e para o aprendizado, permitindo que o usuário verifique o próprio desempenho de seu treinamento para aprender a linguagem de sinais, por meio da resposta da classificação da *selfie image*.

O usuário deste *software* tem como desejo o auxílio na tradução simultânea em uma comunicação com um falante da língua viso-espacial ou ter a curiosidade de poder aprender a linguagem de sinais por meio do aplicativo. Para exercer o desejo do usuário, o aplicativo apresenta uma funcionalidade básica à visão do usuário, que trata de captar uma imagem de um gesto e, após um tempo de execução, o servidor identificará esse sinal e o retorná ao aplicativo a sua respectiva tradução.

Os principais itens identificados foram:

- Realizar a captação de imagens;
- Ter um servidor que receba os dados e possa realizar o processamento nele;
- Realizar a identificação por métodos de processamento de imagem e rede neural artificial;
- Retornar ao usuário uma resposta correta da identificação.

B. *Product Backlog*

Neste *product backlog*, apresentado na Figura 9, são listadas as funcionalidades necessárias para a criação do *software*. Cada funcionalidade apresenta um código, uma descrição e a sua data de inserção no projeto. Ao serem desenvolvidas, deverá ser mencionada a data que a funcionalidade foi selecionada para a implementação.

PRODUCT BACKLOG			
Código	Descrição	Data de Inserção	Data de Seleção
1	Permitir que o usuário tire uma foto com o aplicativo	01/06/2019	12/07/2019
2	O aplicativo deve estar conectado com a internet	01/06/2019	17/07/2019
3	O aplicativo permite a captação de uma foto	01/06/2019	20/07/2019
4	O aplicativo pode enviar uma foto para o servidor	01/06/2019	01/08/2019
5	O aplicativo tem que receber um resultado no fim do processo	01/06/2019	17/11/2019
6	O servidor deve estar preparado para receber a foto	01/06/2019	19/06/2019
7	O servivor deve contrer um banco de dados	01/06/2019	04/07/2019
8	O banco de dados deve conter dados	01/06/2019	02/10/2019
9	O servidor deve contrer uma rede neural artificial	01/06/2019	15/10/2019
10	O servidor deve ter um sistema de processamento de imagens(SPI) de imagem	01/06/2019	06/08/2019
11	O SPI deve conter uma etapa de pré-processamento	02/06/2019	12/08/2019
12	O SPI deve conter uma etapa de segmentação	02/06/2019	19/08/2019
13	O SPI deve fazer a extração de caracteristicas	02/06/2019	26/09/2019
14	O servidor deve treinar a rede neural	02/06/2019	20/10/2019
15	A rede neural deve realizar a classificação	02/06/2019	10/11/2019
16	A rede neural deve retornar ao servidor o resultado da classifição	02/06/2019	15/11/2019
17	O servidor deve fornecer as caracteristicas da imagem para a rede neural	02/06/2019	18/10/2019

Figura 9. Representação do Product Backlog

C. Protótipo do *Software*

Nesta seção é apresentada uma exemplificação visual do aplicativo, representada na Figura 10. Ao abrir o aplicativo, será utilizada a câmera do *smartphone* para possibilitar a visão do que pode ser captado. Assim quando o usuário localizar o falante da língua viso-espacial gesticulando, ele irá captar o gesto. Após o sistema realizar o processamento e reconhecimento da imagem, séra retornado a reposta ao aplicativo que irá informar ao usuário a tradução do gesto.

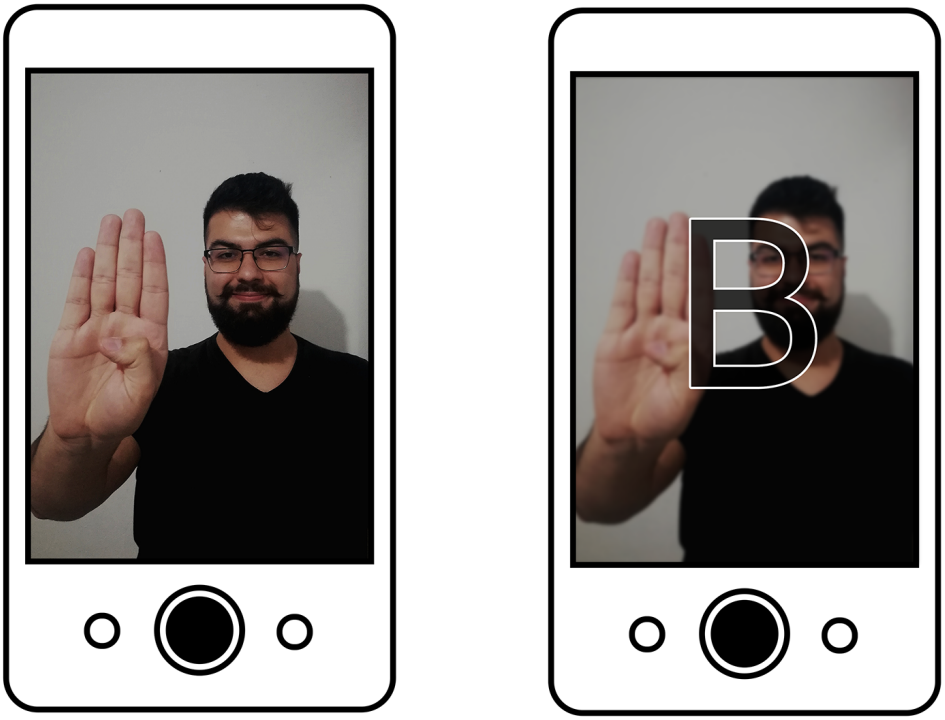

Figura 10. Aplicativo para captar e traduzir a imagem

Apêndice II

Esta fase tem como objetivo desenvolver um conjunto de elementos a partir do elementos gerados na fase anterior. Esses elementos serão o *sprint backlog*, planta de desenvolvimento e produto.

A. *Sprint Backlog*

Esta seção é responsável em organizar as funcionalidades obtidas no *Product Backlog* e assim ordená-las por prioridade. Sendo assim as funcionalidades encontradas no topo da lista apresentam uma maior prioridade para serem implementadas. Na Figura 11 é possível visualizar a tabela dividida entre as determinadas funcionalidades. Assim, apresentando as suas respectivas características: Código, Descrição, Data de Inserção, Tempo de Desenvolvimento contendo os sub-elementos, Orçamento e Realizado.

SPRINT BACKLOG						
Código	Descrição	Data de Inserção	Tempo de construção(h)		Data de validação	Retrabalho
			Orçamento	Realizadas		
6	O servidor deve estar preparado para receber a foto	01/06/2019	8	10	04/07/2019	Sim
7	O servivor deve contrer um banco de dados	01/06/2019	2	8	12/09/2019	Sim
1	Permitir que o usuário tire uma foto com o aplicativo	01/06/2019	4	4	17/07/2019	Não
2	O aplicativo deve estar conectado com a internet	01/06/2019	2	2	20/07/2019	Não
3	O aplicativo permite a captação de uma foto	01/06/2019	4	12	01/08/2019	Sim
4	O aplicativo pode enviar uma foto para o servidor	01/06/2019	2	2	06/08/2019	Não
10	O servidor deve ter um sistema de processamento de imagens(SPI) de imagem	01/06/2019	4	10	12/08/2019	Sim
11	O SPI deve conter uma etapa de pré-processamento	02/06/2019	12	12	19/08/2019	Sim
12	O SPI deve conter uma etapa de segmentação	02/06/2019	30	40	26/09/2019	Sim
13	O SPI deve fazer a extração de caracteristicas	02/06/2019	8	4	02/10/2019	Não
8	O banco de dados deve conter dados	01/06/2019	20	12	15/10/2019	Sim
9	O servidor deve contrer uma rede neural artificial	01/06/2019	8	4	18/10/2019	Não
17	O servidor deve fornecer as caracteristicas da imagem para a rede neural	02/06/2019	4	5	20/10/2019	Sim
14	O servidor deve treinar a rede neural	02/06/2019	30	42	Invalido	Sim
15	A rede neural deve realizar a classificação	02/06/2019	12	4	12/11/2019	Sim
16	A rede neural deve retornar ao servidor o resultado da classifição	02/06/2019	4	1	15/11/2019	Não
5	O aplicativo tem que receber um resultado no fim do processo	01/06/2019	2	1	invalido	Não

Figura 11. Representação do Sprint Backlog

B. Planta de desenvolvimento

Esta seção tem como objetivo apresentar as relações entres os Atores das funcionalidades. Assim será utilizado o Diagrama de Domínio, apresentado na Figura 12, para exibir as relações entre os elementos.

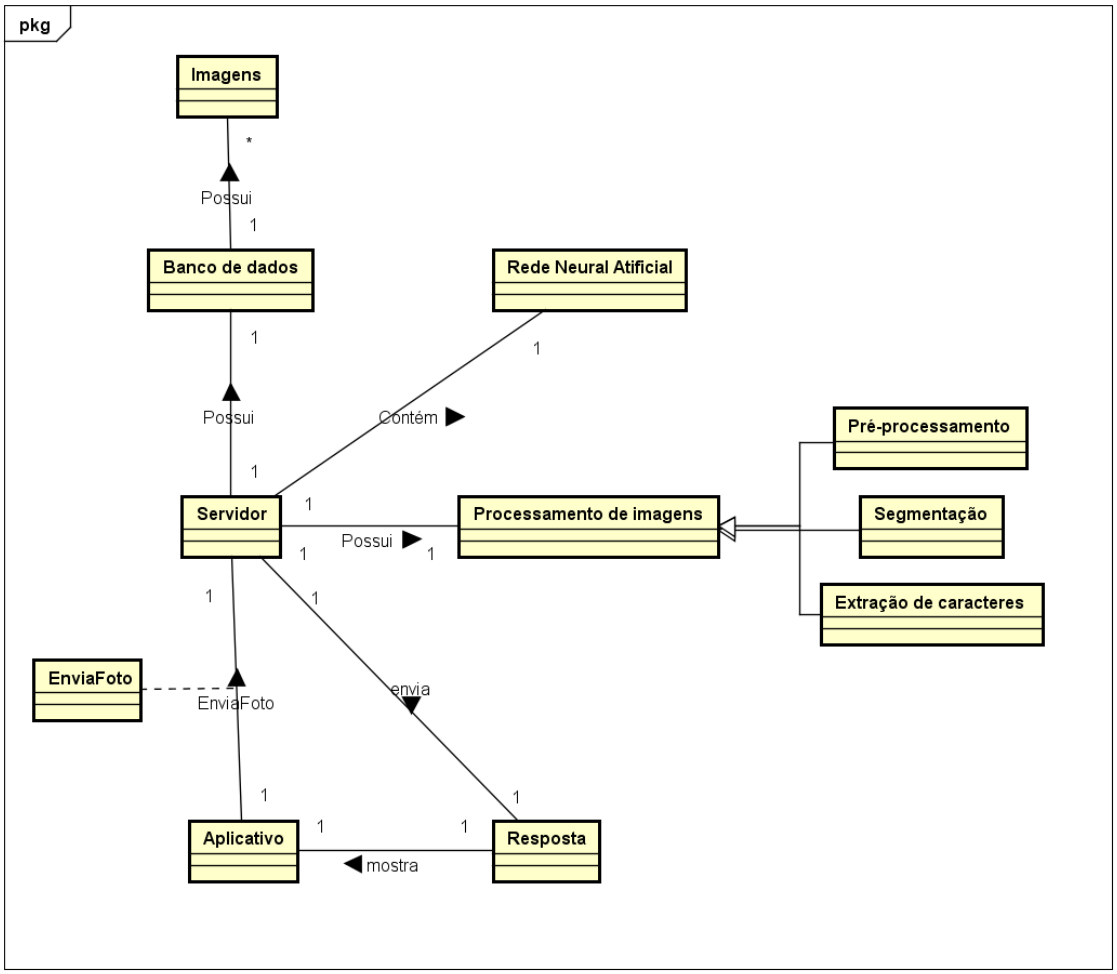

Figura 12. Diagrama de dominio

Apêndice III

Figura que ilustra o local de teste realizado para validar o processamento de imagem.

Figura 13. Cenário de teste

Apêndice IV

Figura que representa o resultado obtido ao realizar o treinamento da RNA utilizando diferentes quantidade de vetores como dados de entrada, mencionada na subseção 6.1 Problemas e dificuldades.

Resolução	Quantidade em vetor	Tempo de interação(ms)
50 x 50	2500	1032,139
70 x 70	4900	3051,696
90 x 90	8100	36528,296
100 x100	10000	38481,733

Figura 14. Tabela contendo o tempo gasto para realizar 20 interações, com diferentes dados de entrada

Apêndice V

Figura que contém diversas informações coletadas dos testes de treinamento da RNA. Nesta tabela, os dados estão ordenados do maior valor de margem de erro até o menor, na qual quanto mais perto do 0, melhor o valor treinado. Apesar do último resultado apresentar um baixo nível de erro, não conseguiu identificar nenhum resultado.

Camadas intermediárias	Neuronios utilizados em cada camada	Margem de erro	Tempo de treinamento(ms)	Sinais identificados
3	221, 221, 221	0,6588000148534780	1469120,180	a, c, d, f, g, l, v
1	2460	0,49919999510049800	1466775,919	a, c, d, g, l
2	700, 700	0,41940000468492500	1459571,906	a, c, d, e
3	1089, 1089, 1089	0,41940000116825100	1466349,055	a, b, d, e
2	1230, 1230	0,3396000048816200	1469943,803	a, b, f
2	221,221	0,3392000047564510	1472120,038	d, f, v
2	1633, 1633	0,3393999141175270	1472878,452	a, b, v
2	2460, 2460	0,25940000896155800	1470501,732	c, i
2	3267, 3267	0,25940000557899500	1474408,187	c, e
3	3267, 3267, 3267	0,1796000016927720	1459060,277	b
3	2460, 2460, 260	0,17960000097753800	1468384,409	d
1	3267	0,1792000094120870	1462199,853	c
4	221, 221, 221, 221	0,099800000756979	1468999,066	-

Figura 15. Testes do treinamento

Apêndice VI

Figura que representa o resultado obtido ao executar a funções que foram implementadas no código do processamento de imagem, mas não estão sendo utilizados para extrair dados. Essas funções podem ser utilizadas em projetos futuros para extrair mais informações, assim tendo a possibilidade de melhorar o treinamento da RNA.

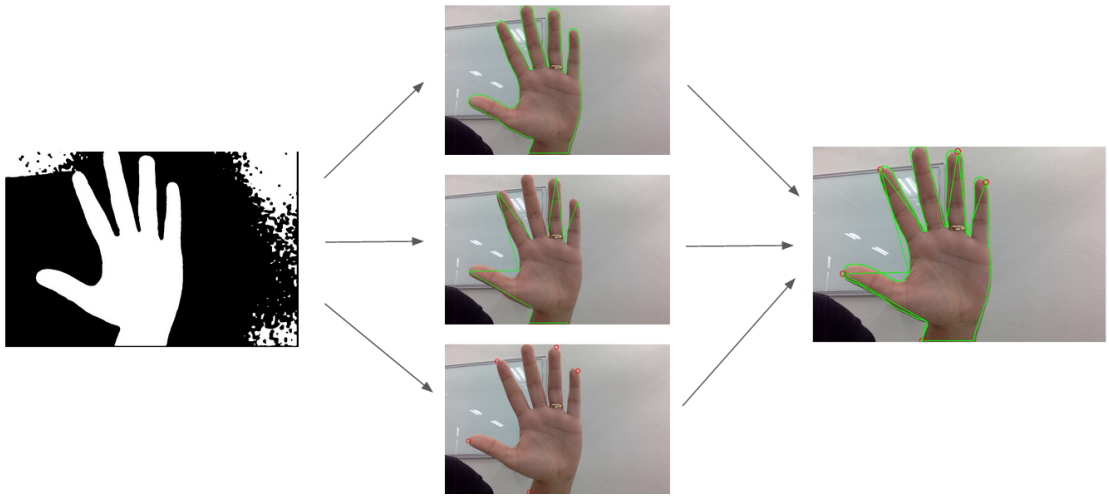

Figura 16. Dados extraídos pelo sistema a partir da máscara binária

Anexo I

Figura que representa o fluxo do processo Scrum Solo, mencionada na seção 4. Metodologia.

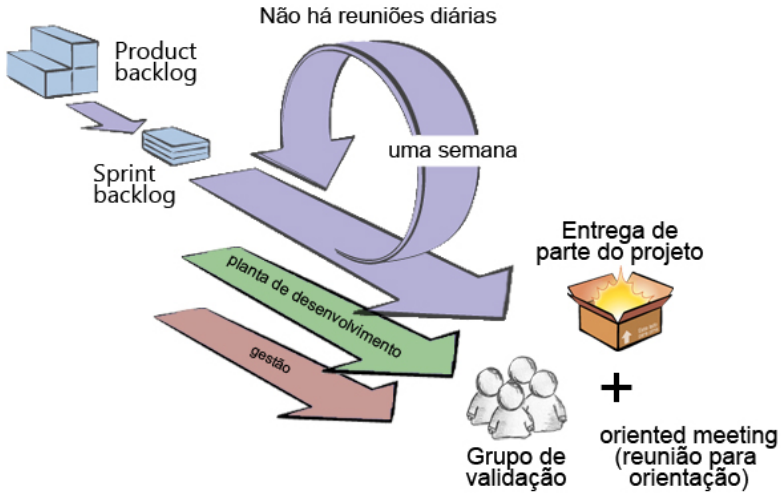

Figura 17. Fluxo do processo Scrum Solo [Pagotto et al. 2016]