FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS

FATEC PROFESSOR JESSEN VIDAL

Felipe Menino Carlos

ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO

São José dos Campos

2019

Felipe Menino Carlos

ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS

FATEC PROFESSOR JESSEN VIDAL

Orientador: Me. Giuliano Araujo Bertoti

São José dos Campos

2019

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Menino Carlos, Felipe
ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO
São José dos Campos, 2019
73f.

Trabalho de Graduação – Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
FATEC de São José dos Campos: Professor Jessen Vidal, 2019
Orientador: Me. Giuliano Araujo Bertoti
Coorientador:

Áreas de Conhecimento. I. Faculdade de Tecnologia. FATEC de São José dos Campos: Professor Jessen Vidal. Divisão de Informação e Documentação. II. ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO

REFERÊNCIA BIBLIGRÁFICA —

Menino Carlos, Felipe. ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO2019. 73f. Trabalho de Graduação – FATEC de São José dos Campos: Professor Jessen Vidal.

CESSÃO DE DIREITOS —

NOME DO AUTOR: Felipe Menino Carlos
TÍTULO DO TRABALHO: ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO
TIPO DO TRABALHO/ANO: Trabalho de Graduação/2019

Felipe Menino Carlos
RG: 50.061.498-2

Felipe Menino Carlos

ICAN.JS: RECURSOS ASSISTIVOS NA WEB UTILIZANDO TÉCNICAS DE APRENDIZADO PROFUNDO

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Composição da Banca

Me. Giuliano Araujo Bertoti
Orientador

Dr. Reinaldo Gen Ichiro Arakaki
Professor Convidado

William Antônio Siqueira
Convidado externo

Luan Rafael Castor Pinheiro
Convidado Externo

São José dos Campos
2019

Dedico este trabalho a meus pais, Jaqueline e Silvano.

Agradecimentos

Agradeço a toda a minha família por me ajudar e incentivar durante toda a jornada de desenvolvimento deste trabalho. Sem vocês, nada disso seria possível.

Agradeço a todos os professores da Faculdade de Tecnologia de São José dos Campos - Professor Jessen Vidal, em especial ao professor Me. Giuliano Bertoti por toda confiança em mim depositada, por todas as conversas e por me mostrar o fantástico mundo da Inteligência Artificial.

Agradeço ao pessoal do Laboratório de Instrumentação de Sistemas Aquáticos (LabISA), que nos últimos dois anos me ajudaram muito no entendimento de tópicos relacionados ao assunto.

Agradeço também aos colegas e amigos que contribuíram de alguma forma para a realização deste Trabalho, em especial: Carlos Neto, Felipe Carvalho, Mauricio Yassunaga e Weslei Luiz.

"Se cheguei até aqui foi porque me apoiei no ombro de gigantes."
(Isaac Newton)

Resumo

Pessoas com qualquer tipo de deficiência tem o direito de uma vida normal, sem nenhum tipo de restrição por conta de suas condições Físico-Mentais. Esse direito garante as pessoas com deficiência uma vida digna. Recursos assistivos representam uma garantia de dignidade e acessibilidade à vida de pessoas com deficiência. Cada um desses recursos assistivos são criados para potencializar as características já presentes nos indivíduos, diminuindo as barreiras existentes por características que não possuem. No entanto, os recursos assistivos devem ser adaptados para cada contexto de aplicação, o que pode ser um problema, quando cada contexto de aplicação possui um preço de aquisição diferente. Com os recursos tecnológicos atuais é possível criar recursos assistivos adaptáveis a variados contextos, desta forma, o objetivo deste Trabalho é desenvolver recursos assistivos para *web* utilizando técnicas de Aprendizado Profundo. Para isso os modelos de redes neurais convolucionais MobileNet e PoseNet foram utilizados no desenvolvimento dos recursos assistivos e sua aplicação em páginas *web* apresentaram bons resultados na generalização, facilidade de uso e aplicabilidade.

Palavras-chave: Recursos assistivos, Aprendizado Profundo, Redes Neurais, Redes Neurais Convolucionais, JavaScript

Abstract

People with any kind of disability have the right to a normal life, without any kind of restriction because of their Physical-Mental conditions. This right guarantees people with disabilities a decent life. Assistive resources represent a guarantee of dignity and accessibility to the lives of people with disabilities. Each of these assistive features is created to enhance the characteristics already present in individuals, reducing the barriers by characteristics they do not have. However, assistive features must be tailored to each application context, which can be a problem, when each application context has a different acquisition price. With the current technological resources, it is possible to create assistive resources adaptable to various contexts, in this way, the objective of this work is to develop assistive web resources using Deep Learning techniques. For this, the convolutional neural network models MobileNet and PoseNet were used in the development of the assistive resources and their application in web pages presented good results in generalization, ease of use and applicability.

Keywords: Assistive Resources, Deep Learning, Neural Network, Convolutional Neural Network, JavaScript

Lista de ilustrações

Lista de abreviaturas e siglas

RNA	*Redes Neurais Artificiais*
PMC	*Perceptron Multicamadas*
AM	*Aprendizado de Máquina*
AP	*Aprendizado Profundo*
TFJS	*TensorFlow.js*
RNC	*Rede Neural Convolucional*
IBGE	*Instituto Brasileiro de Geografia e Estatística*
H5	*Arquivo Hierárquico do Keras*
API	*Application Programming Interface*
TA	*Tecnologia assistiva*
RA	*Recurso assistivo*
Libras	*Língua Brasileira de Sinais*
Colab	*Colaboratory*
GPU	*Graphics Processing Unit*

Sumário

1 Introdução

Este capítulo apresenta a motivação para o desenvolvimento deste trabalho, os objetivos deste e a metodologia adotada.

1.1 Motivação

De acordo com o censo do Instituto Brasileiro de Geografia e Estatística (IBGE), realizado em 2010, no Brasil, há cerca de 45 milhões de pessoas com algum tipo de deficiência e todas estas necessitam de uma vida independente e inclusiva, não sendo impedidas de realizar suas atividades por conta da deficiência.

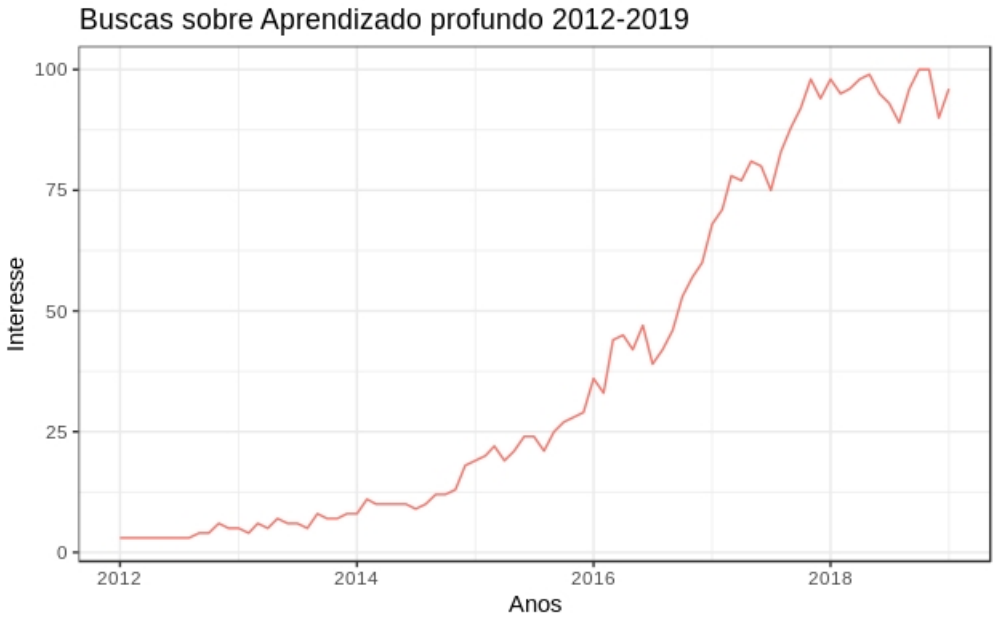
Para uma parte deste público-alvo, a solução para a inclusão e independência são as tecnologias assistivas (ANDRIOLI, 2017), que através dos avanços tecnológicos criam possibilidades para as pessoas que possuem deficiência (BERSCH, 2017). Este tipo de tecnologia deixa de lado as deficiências e foca apenas nas habilidades já presentes nos indivíduos (GUILHERMANO et al., 2016).

Porém no Brasil, mesmo com diferentes iniciativas surgindo ao longo dos anos (BERSCH, 2017) há uma grande dificuldade de acesso aos recursos de tecnologia assistiva (TA) (ANDRIOLI, 2017), causados por diversos fatores, a citar, o alto custo, por conta da grande necessidade da criação de equipamentos específicos para cada caso (GUILHERMANO et al., 2016), e também a necessidade de importação destes equipamentos (ANDRIOLI, 2017).

Por outro lado, tem-se técnicas de Aprendizado Profundo (AP), que através da aplicação de redes neurais artificiais (RNA) profundas, desde 2012 vem sendo apresentadas como o estado-da-arte para a solução dos mais diversos problemas, nos mais variados contextos (PONTI; COSTA, 2018; PRESS, 2019), isto ocorrendo principalmente por conta da alta capacidade de generalização e adaptabilidade desses algoritmos (SILVA, 2017; PONTI; COSTA, 2018; PRESS, 2019).

Essas características fizeram com que a procura e aplicação desses algoritmos aumentasse exponencialmente, vide Figura 1. Isso fez com que diversas empresas como Google e Facebook começassem a utilizar muito dessas técnicas em seus sistemas e aplicações, o que impulsionou a área de uma forma nunca antes vista.

Figura 1 – Interesse em Aprendizado Profundo de 2012 à 2019



Fonte – Google (2019)

Com empresas interessadas na rápida prototipação e implementação dessas técnicas em seus serviços, diversas bibliotecas de códigos, nas mais variadas linguagens, que facilitam a aplicação dessas técnicas foram criadas e muitas disponibilizadas para o público em geral. Dentre as bibliotecas disponíveis, destaca-se Paszke et al. (2017), Smilkov et al. (2019), Abadi et al. (2015) e Chollet (2015). Estas bibliotecas trouxeram benefícios não apenas para as empresas por trás delas, mas também para todas as comunidades com interesse em tais técnicas.

Para a área científica, por exemplo, veio a oportunidade de aplicação de tais técnicas para impulsionar diferentes estudos como em Frid-Adar et al. (2018) e Vargas, Paes e Vasconcelos (2016), além de outras empresas que se beneficiaram expandindo seus negócios e áreas de atuação.

Neste contexto, para o desenvolvimento de recursos assistivos (RAs) mais precisos e acessivos, as técnicas de AP começaram a ser aplicadas, como é o caso de Magalhães (2018), que desenvolve um identificador de gestos de Libras estado-da-arte com tais técnicas. Mas, essas aplicações ainda estão muito restritas ao desenvolvimento e geração de modelos de redes neurais profundas, fazendo com que seja necessário a implementação desses em sistemas assistivos, como a biblioteca de navegação apresentada por Ramos (2019). Assim, este Trabalho é motivado pela possibilidade do desenvolvimento de RAs para páginas *web*, aplicando técnicas de AP, para trazer mais acessibilidade e precisão às ferramentas desenvolvidas.

1.2 Objetivo Geral

O objetivo geral deste Trabalho é desenvolver uma biblioteca JavaScript que possa levar recursos assistivos desenvolvidos através de técnicas de Aprendizado Profundo para páginas *web*.

1.3 Objetivo Específico

Para a consecução do objetivo geral foram estabelecidos os seguintes objetivos específicos:

- Desenvolvimento de um recurso assistivo que permite a movimentação do *cursor* do *mouse* em páginas *web* através de gestos da cabeça;
- Desenvolvimento de um recurso assistivo que permite a escrita de texto em campos de páginas da *web* através de gestos de Libras;
- Integração das ferramentas desenvolvidas em uma biblioteca JavaScript.

1.4 Metodologia

A realização dos objetivos específicos deste Trabalho são feitas através da aplicação de modelos de AP no desenvolvimento dos RAs para páginas da *web*, todos estes consolidados através de uma biblioteca JavaScript, esta nomeada de ICan.js.

Para o desenvolvimento do RA de escrita de texto com gestos de Libras, dados são coletados através de diferentes usuários através de um sistema de coleta, isso para que um modelo MobileNet (HOWARD et al., 2017) seja treinado e possibilite o reconhecimento de gestos, para então traduzir esses em textos. Já no RA para a movimentação do *cursor* do *mouse*, o modelo PoseNet (OVED, 2018) junto a regressões lineares são aplicados para o mapeamento dos gestos do usuário em posições do *cursor* na tela.

A disponibilização e utilização dos modelos de AP na biblioteca desenvolvida neste Trabalho, é feita través do módulo TensorFlow para JavaScript (SMILKOV et al., 2019).

1.5 Organização do trabalho

Este Trabalho está organizado nos seguintes capítulos:

- 2 - Fundamentação Teórica: O capítulo expõe os conceitos necessários para a compreensão do presente trabalho;

- 3 - Desenvolvimento: Este capítulo aborda detalhadamente a especificação da biblioteca JavaScript, assim como a metodologia empregada para o desenvolvimento de cada um dos recursos assistivos;
- 4 - Resultados: Neste capítulo são expostos, através de aplicações de exemplo, os resultados alcançados com a metodologia aplicada na implementação dos recursos assistivos e na biblioteca;
- 5 - Considerações Finais: Este capítulo apresenta as conclusões obtidas com os resultados, assim como uma breve sugestão de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo serão fundamentados os conhecimentos básicos para o entendimento do trabalho.

2.1 Recursos assistivos

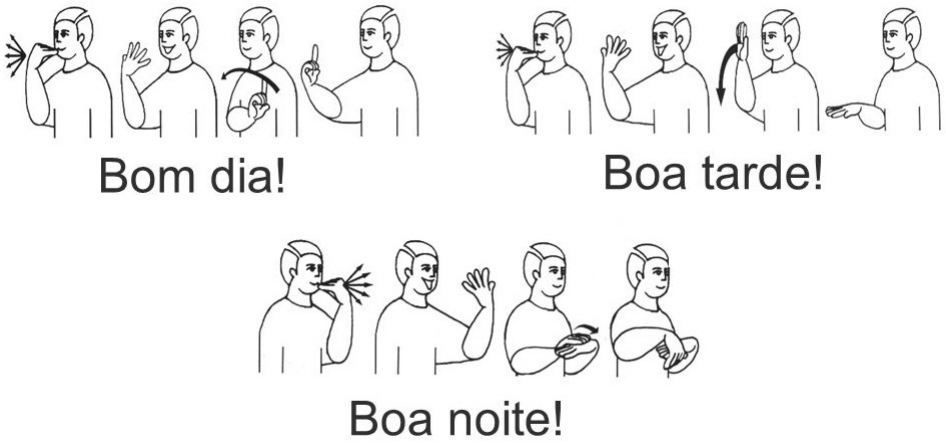
No Brasil, pessoas que possuem algum tipo de deficiência tem o direito de receber tratamento especial em diferentes meios e serviços (ANDRIOLI, 2017), para que a dignidade e necessidades básicas do individuo sejam garantidas (Oliveira; CARDOSO, 2011). Uma das formas de garantir este direito a pessoas com deficiência é através da aplicação de Tecnologias Assistivas.

Tecnologia assistiva é o termo que identifica todo o arsenal de recursos e serviços assistivos (BERSCH, 2017), sendo, os recursos assistivos tudo aquilo que é produzido para aumentar, manter ou melhorar as capacidades de pessoas com deficiência (TONOLLI; BERSCH, 2017) e os serviços representam tudo o que ajuda na aplicação e utilização de recursos assistivos (TONOLLI; BERSCH, 2017).

De acordo com Tonolli e Bersch (2017) existem diferentes tipos de TAs, sendo algumas delas: recursos de acessibilidade ao computador, auxílios para surdos ou com déficit auditivo e auxílios para a vida diária.

Todos esses recursos podem ser construídos através da aplicação de diferentes técnicas e métodos. Por exemplo, para o caso de uma pessoa que possua deficiência auditiva e necessita de um atendimento especial para seu processo de aprendizagem, é possível criar recursos assistivos que facilitem o aprendizado e comunicação através de pranchas de comunicação (BERSCH, 2017). Também pode ser aplicado, formas de comunicação através da Língua Brasileira de Sinais (Libras), uma língua manuo-motora de recepção visual, ou seja, através de sinais e gestos criam-se representações de letras e palavras (Figura 2), sendo a mais usada pelos surdos no Brasil (CARVALHO, 2007) e é reconhecida oficialmente nos meios legais segundo a Lei n 10.436, de 25 de Abril de 2002. Além disso, seu uso em RAs pode ser feito para acessibilidade, uma vez que, boa parte dos surdos não são alfabetizados na Língua Portuguesa, somente em Libras (GUILHERMANO et al., 2016).

Figura 2 – Exemplos de sinais de Libras



Fonte – Markewicz (2017)

Os exemplos de RAs listados acima podem ser recursos desenvolvidos para *tablets*, celulares e aplicações *web* e não somente, em certos casos, simples representações em papel, ou mesmo em um brinquedo (BERSCH, 2017), podem ser considerados RAs.

2.2 Regressão linear

Análise de regressão é uma técnica preditiva (TAN; STEINBACH; KUMAR, 2009) que tem com objetivo modelar a relação funcional entre uma variável dependente e uma ou mais variáveis independentes (PETERNELLI, 2003). Assim, verifica-se a variação de uma variável, decorrente a variação de outra variável (PETERNELLI, 2003; MANN, 2006).

Essa relação funcional pode ser apresentada de diversas formas: linear, quadrática, cúbica, dentre outras (PETERNELLI, 2003). A representação de cada um desses comportamentos é feita através de modelos matemáticos (MANN, 2006).

A regressão linear, por exemplo, é a representação de uma relação funcional linear entre duas variáveis, sendo essas, a variável dependente, que representa a variável que está sendo explicada e as variáveis independentes que explicam a variável independente (MANN, 2006).

Uma das formas de representação deste modelo é apresentado na Equação 2.1.

$$\widehat{y} = a + bx \tag{2.1}$$

Onde, $\widehat{y}$ é a variável dependente, que está sendo explicada, x a variável independente, a é o termo constante e b é a inclinação. Sendo que, x e y representam os coeficientes

de regressão, que são modificados para que a relação funcional entre as variáveis sejam explicadas (TAN; STEINBACH; KUMAR, 2009).

No momento em que as coeficientes de regressão são ajustados e fazem a representação do comportamento entre duas ou mais variáveis, como citado acima, passa a ser possível a sua aplicação para a predição de novos valores (TAN; STEINBACH; KUMAR, 2009), esses que ao serem preditos terão o mesmo padrão de comportamento do conjunto de dados aplicado para a criação dos coeficientes.

2.3 Inteligência artificial

Sistemas inteligentes de forma geral são aqueles que apresentam a capacidade de planejar e resolver problemas através de dedução e indução utilizando conhecimentos de situações anteriores (ZUBEN, 2013), e a inteligência artificial, é um campo da ciência e engenharia de computação (ZUBEN, 2013) que possibilitam a sistemas computacionais, perceber, raciocinar e agir (WINSTON, 1992).

As técnicas computacionais mais utilizadas para o desenvolvimento e aplicação de inteligência artificial, são aquelas relacionadas ao aprendizado de máquina. Esta que é uma área que tem como objetivo principal, desenvolver técnicas que permitam aos sistemas adquirir conhecimento de forma automática e com estes conhecimentos tomar decisões (BARANAUSKAS, 2007).

Para a realização do aprendizado de máquina, existem diversas técnicas, que vão de simples regressões estatísticas, até modelos complexos, como às redes neurais artificiais (NG; KATANFOROOSH; MOURRI, 2016).

2.4 Redes neurais artificiais

Redes neurais artificiais são sistemas computacionais que busca modelar o sistema cerebral natural humano, estas que são uma das formas de soluções de problemas apresentados dentro do âmbito de inteligência computacional (CINTRA, 2019).

Por buscar modelar o cérebro humano, as RNAs utilizam como unidade básica de processamento, os neurônios artificiais (HAYKIN, 2001), da mesma forma que o cérebro utiliza os neurônios biológicos.

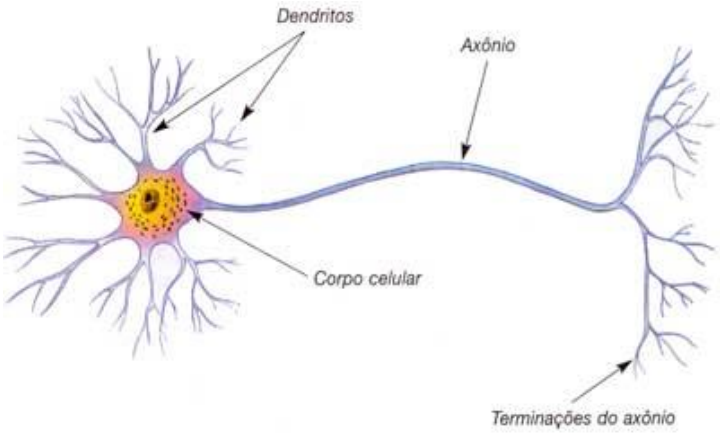
2.4.1 Neurônio biológico

Todo o processamento de informações no cérebro humano, é feito através de elementos biológicos de processamento, que operam em paralelo para a produção de ações apropriadas para cada estímulo recebido pelo corpo. A célula base do sistema nervoso

cerebral é o neurônio (Figura 3), e sua principal função é conduzir impulsos (Representando os estímulos) levando em consideração as condições do corpo e assim produzindo ações. Os neurônios também são os responsáveis pelos atos do pensamento e armazenamento de informações (SILVA; SPATTI; FLAUZINO, 2016).

Os neurônios podem ser divididos em três partes elementares, os dendritos, que captam de forma continua os impulsos vindos de outros neurônios, o corpo celular, que processa todas as informações captadas e os axônios que enviam as informações processadas no corpo celular para outros neurônios.

Figura 3 – Ilustração do neurônio biológico



Fonte – Remes (2016)

Estima-se que a rede neural cerebral, possui cerca de 100 bilhões de neurônios, cada um desses mantendo conexão com uma média de 6.000 outros neurônios, gerando cerca de 600 trilhões de conexões (SHEPHERD, 1990). A região de conexão entre os neurônios são chamadas de sinapses, que exercem ponderações entre as conexões entre os neurônios, fazendo assim com que haja regiões do cérebro especializadas em diferentes tarefas.

A representação inicial deste conjunto de neurônios em sistemas de computação foi implementada através de circuitos eletrônicos, apresentado por McCulloch e Pitts (1943), estes que foram utilizados como base para a criação dos modelos de neurônios artificiais apresentados por Hodgkin e Huxley (1952).

2.4.2 Neurônio artificial

Como citado anteriormente, os neurônios artificiais, são modelos computacionais para a representação do neurônio biológico nas RNAs, e da mesma forma que em um neurônio biológico, a representação deste é feita com três elementos básicos (HAYKIN, 2001):

- Conjunto de sinapses, cada uma caracterizada por um peso, este que indica a relevância de cada valor de entrada;
- Somador, ou combinador linear, que faz a ponderação dos valores de entrada com as respectivas sinapses do neurônio;
- Função de ativação utilizada para restringir os valores de saída do neurônio.

Ainda de acordo com Haykin (2001), a estes modelos neuronais pode-se aplicar um *bias*, este que será o responsável pelo aumento ou diminuição dos valores de entrada da função de ativação. Em termos matemáticos, pode-se descrever um neurônio k (Figura 4) com as seguintes equações (HAYKIN, 2001):

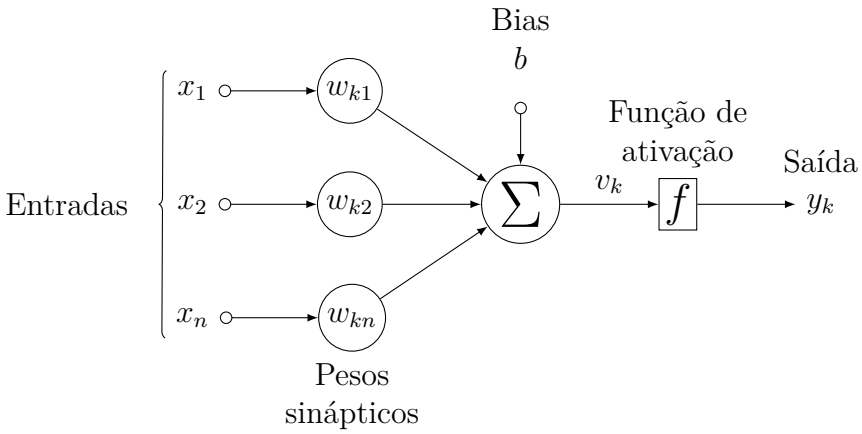
$$u_k = \sum_{j=1}^{n} w_{kj} x_j \tag{2.2}$$

e

$$y_k = f(u_k + b_k) \tag{2.3}$$

onde $x_1, x_2, ..., x_n$ são os sinais de entrada; $w_{k1}, w_{k2}, ..., w_{kn}$ são os pesos sinápticos do neurônio k; u_k é a saída do combinador linear; b_k é o *bias*; $f(u_k + b_k)$ a função de ativação; e y_k representa a saída do neurônio.

Figura 4 – Neurônio Artificial



Fonte – Adaptado de Haykin (2001)

A partir da Figura 4 é possível realizar uma comparação entre cada um dos elementos do neurônio artificial e biológico. Os sinais de entrada, advindos do meio externo, normalmente uma aplicação, são análogos aos impulsos elétricos captados pelos dendritos no neurônio biológico. Os pesos sinápticos representam a importância do sinal recebido para o neurônio, o que representa as ponderações exercidas pelas junções sinápticas do modelo biológico, ou seja, a força do caminho entre as sinapses, citados anteriormente. O

campo de somatório junto a função de ativação, representam o corpo celular do neurônio biológico, é nesta parte que os resultados criados pelo neurônio são calculados (SILVA; SPATTI; FLAUZINO, 2016)

2.4.3 Arquiteturas de rede

Para Silva, Spatti e Flauzino (2016) uma RNA pode ser constituída de até três partes diferentes, estas denominadas de camadas, as quais são nomeadas a seguir:

- Camada de entrada: É a camada responsável pelo recebimento de dados;
- Camadas escondidas, intermediárias ou ocultas: São camadas compostas de neurônios responsáveis pela extração de características associadas ao processo ou sistema;
- Camadas de saída: Também constituída de neurônios, esta camada é responsável pela produção e apresentação dos resultados finais da rede.

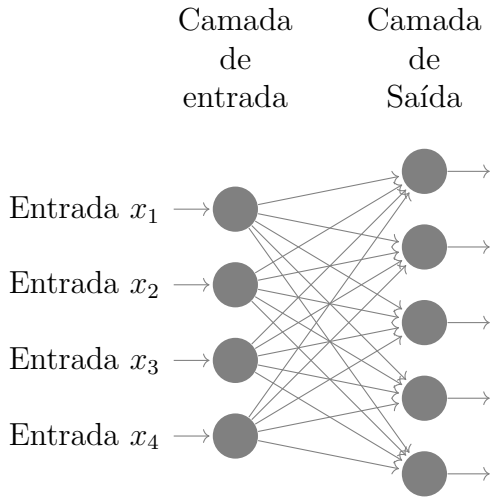
Das camadas descritas acima, devem estar presentes em uma RNA no mínimo a camada de entrada e a camada de saída (CINTRA, 2019).

Diferentes formas de organização de cada uma destas camadas, especialmente relacionadas a forma de relação entre os neurônios, definem as arquiteturas de RNA (SILVA; SPATTI; FLAUZINO, 2016). Haykin (2001) define a existência de duas classes de arquiteturas fundamentais, sendo elas: Redes de alimentação direta, com uma ou várias camadas e Redes recorrentes.

As redes de alimentação direta, são denominadas desta forma por conta de seu fluxo percorrer uma única direção (CINTRA, 2019), iniciando o fluxo na camada de entrada seguindo pelas diferentes camadas até o neurônio de saída. Este tipo de rede pode possuir uma ou várias camadas ocultas.

Para as redes de alimentação direta com uma única camada, tem-se como tipo comum a *Perceptron* (Figura 5). Em sua estrutura são apresentadas duas camadas, entrada e saída, porém são nomeadas de camada única já que existem operações matemáticas ocorrendo apenas na camada de saída (SILVA; SPATTI; FLAUZINO, 2016).

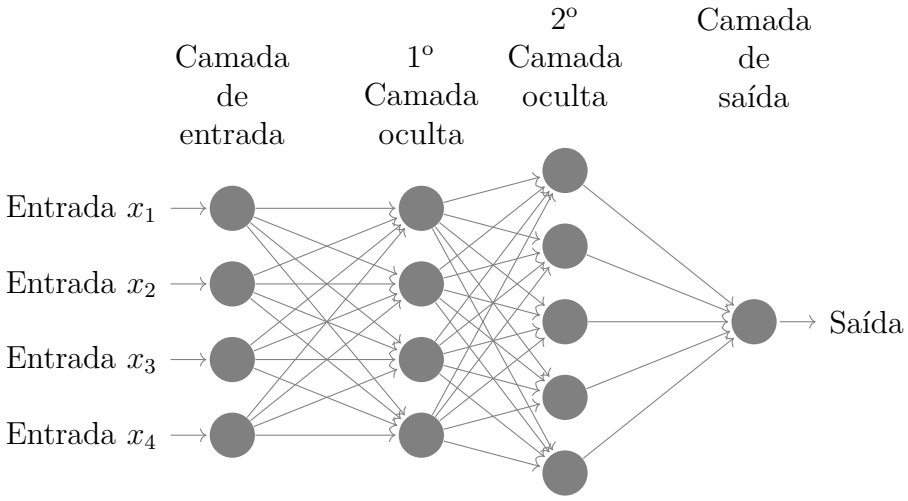
Figura 5 – Rede de alimentação direta de camada única



Fonte – Adaptado de Silva, Spatti e Flauzino (2016)

Já as redes de alimentação direta com múltiplas camadas (Figura 6), diferentes das redes de camada única, possuem diversas camadas ocultas. Para essa classe o tipo mais comum é o Perceptron multicamadas (PMC), que por possuírem mais camadas são capazes de extrair quantidades maiores de características do problema que está sendo modelado pela rede.

Figura 6 – Rede de múltiplas camadas



Fonte – Adaptado de Silva, Spatti e Flauzino (2016)

Por fim, as redes recorrentes que recebem este nome por conta da realimentação entre os neurônios da mesma camada, ou seja, a saída de um neurônio em uma camada, pode servir como entrada para outro neurônio da mesma camada (NELSON, 2017).

Estas formas de organização presentes nas arquiteturas, estão intimamente relacionadas ao processo de aprendizado que é aplicado nas RNAs (HAYKIN, 2001).

2.4.4 Processo de aprendizado

Um dos pontos mais relevantes das RNAs é a generalização (SILVA; SPATTI; FLAUZINO, 2016), onde treina-se levando em consideração um conjunto amostral A, que faz uma boa representação do problema resolvido na tarefa e então após este processo a rede consegue realizar a tarefa não somente para o conjunto A, mas também para um conjunto C qualquer (SILVA; SPATTI; FLAUZINO, 2016).

Porém para a generalização, como citado, é necessário um processo de treinamento, que seja adequado a arquitetura de RNA. Silva, Spatti e Flauzino (2016) definem processo de treinamento como um algoritmo que, através de passos bem definidos ajustam os pesos sinápticos da RNA com o objetivo de permitir o mapeamento das relações dos dados e então generalizar.

Os processos de treinamento podem adotar diferentes estratégias para ensinar as RNAs, e cada estratégia gera um algoritmo de aprendizado diferente, sendo os principais, algorítimos de aprendizado supervisionado e não-supervisionado.

No aprendizado supervisionado, há rótulos $y^{(t)}$ que indicam o comportamento $\widehat{y}$ que a rede deve apresentar para cada $x^{(t)}$ presente em um conjunto de dados $\{(x^{(t)}, y^{(t)}) : 1 \leqslant t \leqslant T\}$ (BEZERRA, 2016). Desta forma, de acordo com os resultados apresentados, ajustes são feitos nos pesos sinápticos e limiares dos neurônios da RNA (SILVA; SPATTI; FLAUZINO, 2016), para que o resultado gerado seja o mais próximo possível de $y^{(t)}$ (OSÓRIO, 1999). Assim, o objetivo do aprendizado supervisionado é gerar para uma entrada x, um valor de y próximo a $y^{(t)}$, sendo que, para valores categóricos de y, tem-se uma classificação e para valores numéricos uma regressão (MURPHY, 2012).

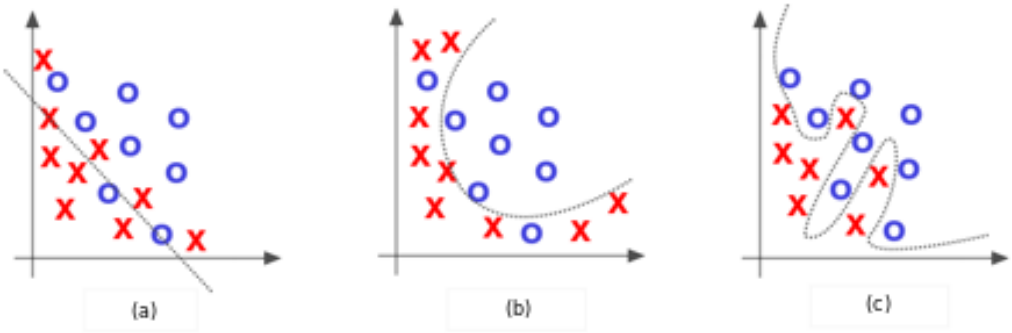
Já o aprendizado não supervisionado, são utilizados apenas os dados, sem qualquer tipo de rótulo, sendo utilizados para gerar grupos (SILVA, 2017) e com isto descobrir novos padrões no conjunto de dados (MURPHY, 2012)

Após a aplicação dos algoritmos de treinamento é necessário avaliar a performance do modelo de RNA gerado, para garantir que o mesmo está sendo capaz de generalizar. Uma maneira de realizar a avaliação é verificando a acurácia da RNA através da quantidade de acertos do modelo frente a um conjunto de dados não apresentado para a RNA durante o processo de treinamento, fazendo com que o conjunto de dados tenha de ser separado em dados de treino e teste (GOODFELLOW; BENGIO; COURVILLE, 2016).

A separação dos dados, pode ser utilizada também para a identificação de problemas de *Underfitting*, que é causado quando o modelo não consegue extrair características

relevantes do conjunto de dados, obtendo altas taxas de erro durante o treinamento, ou mesmo o *Overfitting*, que ocorre quando o modelo extrair características além do necessário do conjunto de dados, fazendo assim com que o modelo tenha baixas taxas de erro no treino e praticamente não acerte no teste (GOODFELLOW; BENGIO; COURVILLE, 2016). Na Figura 7 os problemas expostos anteriormente são representados, onde (a) representa o *Underfitting*, (c) *Overfitting* e (b) o ideal.

Figura 7 – Representação gráfica do *Overfitting* e *Underfitting*



Fonte – Adaptado de Gibson e Patterson (2017)

Para muitos casos as RNAs são utilizadas pois conseguem chegar ao ideal (Figura 7 (b)) (GOODFELLOW; BENGIO; COURVILLE, 2016), garantindo assim a generalização, porém a depender do escopo do problema, as RNAs necessitam de mais camadas, cada uma delas com certas especialidades, para possibilitar a extração mais sofisticada de características do problema modelado pela RNA.

2.5 Aprendizado Profundo

O Aprendizado Profundo (AP) apresenta uma abordagem diferente para os problemas resolvidos com técnicas de RNA, onde múltiplas camadas são empregadas nas arquiteturas, permitindo assim que problemas mais complexos e sofisticados sejam mapeados (GOODFELLOW; BENGIO; COURVILLE, 2016).

As características dos algoritmos de AP fizeram com que estes chegassem ao estado-da-arte em diversos casos, como em Shankar et al. (2017) e Krizhevsky, Sutskever e Hinton (2012).

2.5.1 Redes Neurais Convolucionais

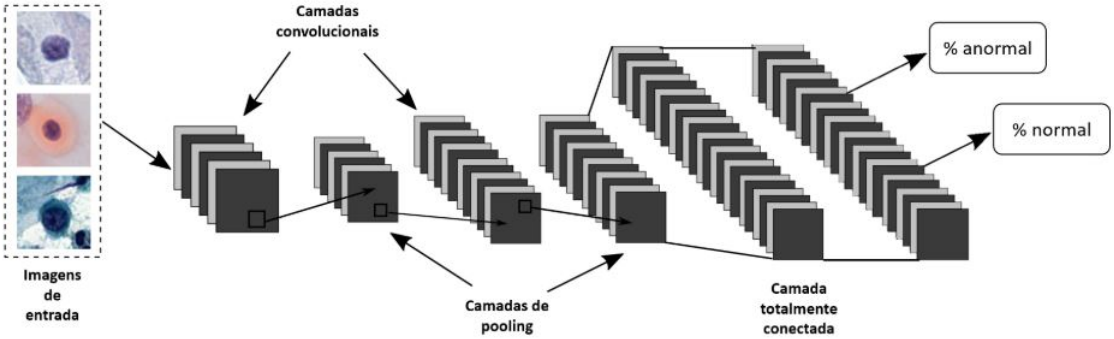
Redes Neurais Convolucionais (RNC) são uma variação das redes PMC, tendo sua criação inspirada no processo biológico de processamento de dados visuais (VARGAS; PAES;

VASCONCELOS, 2016). Estas arquiteturas de AP, são capazes de subdividir os dados para tentar extrair características relevantes a classificação, reduzindo assim o números de parâmetros que deverão ser ajustados pela rede (MIYAZAKI, 2017), melhorando o processo de treinamento (MIYAZAKI, 2017).

As RNCs são utilizadas em problemas em que os dados tem estruturas de grade, como por exemplo, processamento de fala e entendimento da linguagem natural (Uma dimensão, convolução temporal) (MIYAZAKI, 2017) e segmentação e classificação de imagens (Duas dimensões, convolução espacial) (MIYAZAKI, 2017; GOODFELLOW; BENGIO; COURVILLE, 2016).

Um dos primeiros modelos de RNCs propostos foi a LeNet (LECUN et al., 1998) (Figura 8), formada por conjunto de camadas, também nomeada de blocos de funções, onde cada uma delas possui um objetivo específico (ARAÚJO et al., 2017).

Figura 8 – Estrutura básica de RNC proposta por LeCun et al. (1998) aplicado na identificação de tumores normais e anormais



Fonte – Araújo et al. (2017)

As três camadas fundamentais para uma RNC, apresentadas por LeCun et al. (1998), e ilustradas na Figura 8, são as seguintes: convolucional, de *pooling* e totalmente conectada.

2.5.1.1 Camada convolucional

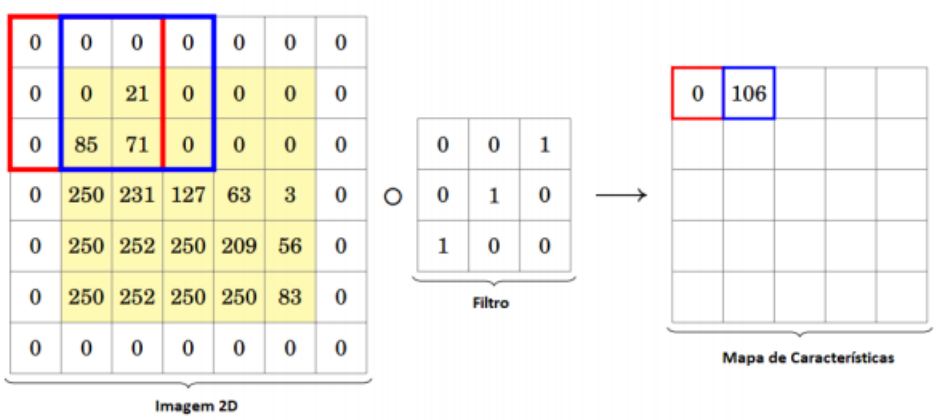
A camada convolucional representa um conjunto de filtros não lineares que percorrerem os dados de entrada sequencialmente e produzem os mapas de características (MIYAZAKI, 2017). O processo de percorrer os dados de entrada ocorre através de passos (*strides*), passando de *pixel* em *pixel* nas imagens de entrada.

Nesta camada há dois conceitos centrais: o campo receptor local e o compartilhamento de pesos sinápticos. Assim, com o campo receptor local, ao contrário das RNAs

tradicionais, na qual todo o volume de entrada é conectado a camada oculta, apenas uma região específica definida pelo compartilhamento de pesos sinápticos é retirada da camada de entrada (NIELSEN, 2018).

A Figura 9 mostra o processo de convolução, onde, um filtro *3*x*3*, definido pelo compartilhamento dos pesos sinápticos, passa sobre uma região dos dados, com passo igual a *1*, e a multiplicação entre eles é realizada, e os valores resultantes são somados e colocados no mapa de características.

Figura 9 – Processo de convolução



Fonte – Adaptado de Pavlovsky (2017)

Com isto, no mapa de características há apenas características relevantes para a classificação.

2.5.1.2 Rectified Linear Units

Ao final da camada de convolução, normalmente aplica-se uma função de ativação (ARAÚJO et al., 2017), e a mais comumente utilizada para RNCs é a *Rectified Linear Units* (NAIR; HINTON, 2010; KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Essa função é calculada pela Equação 2.4.

$$f(x) = max(0, x) \tag{2.4}$$

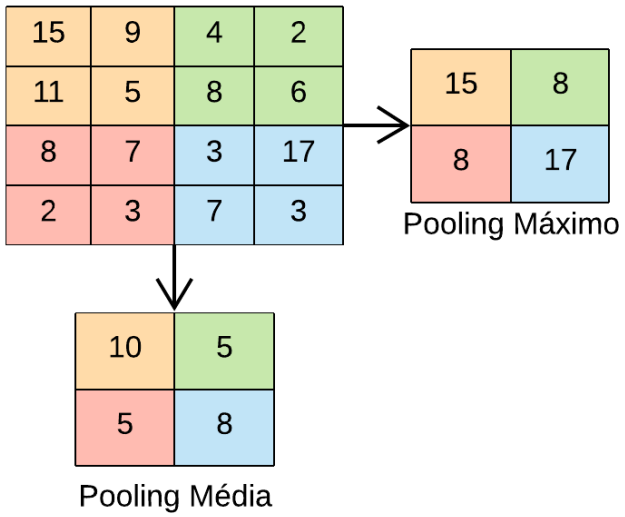
Esta função faz com que, sejam passados para as camadas subsequentes a ela, apenas valores positivos, já que todos os negativos são substituídos por zero.

2.5.1.3 Camada de Pooling

A camada de *Pooling*, comumente utilizada após uma certa camada de convolução (VARGAS; PAES; VASCONCELOS, 2016), tem a finalidade de reduzir a dimensionalidade dos dados do mapa de características (VARGAS; PAES; VASCONCELOS, 2016). Esta redução é feita principalmente para agilizar o processo de treinamento (VARGAS; PAES; VASCONCELOS, 2016).

Esta redução é feita com o agrupamento de valores através de uma janela MxN que é passada pelo mapa de características, aplicando uma função, normalmente de média ou de valor máximo (AMIDI, 2018). A Figura 10 apresenta uma operação de *Pooling* feita através de um filtro 2x2, com a função de valor máximo.

Figura 10 – Processo de *Pooling*



Fonte – Adaptado de Rawat e Wang (2017)

Segundo Nielsen (2018), o filtro de *Pooling* normalmente é utilizado as dimensões 2x2 com um passo de deslocamento igual a 1.

2.5.1.4 Camada totalmente conectada

A saída das camadas convolucionais e de *Pooling* geram as características extraídas dos dados de entrada (ARAÚJO et al., 2017). As camadas totalmente conectadas utilizam estas características para classificar os dados. As camadas totalmente conectadas representam uma PMC (HAYKIN, 2001) descrita nos capítulos anteriores, mas, sua saída é

controlada por uma função de ativação, normalmente *softmax*, responsável pela classificação dos resultados (BISHOP, 2006).

2.5.1.5 Transferência de Aprendizado

Realizar treinamentos de RNCs requer grandes quantidades de dados, não sendo comum realizar treinamento deste tipo de rede do zero (ARAÚJO et al., 2017), normalmente essas redes apresentam muitas camadas e realizar o ajuste de cada camada pode exigir muitos dados. Desta forma é comum utilizar modelos que já possuem parâmetros ajustados para outros conjuntos de dados (PONTI; COSTA, 2018), ou seja, aplica-se um domínio geral em um domínio específico. Todo esse processo é nomeado de Transferência de Aprendizado.

De acordo com Ponti e Costa (2018) existem diversas abordagens para a realização da transferência de aprendizado, sendo algumas delas: (i) Permitir que o algoritmo de treino ajuste todos os pesos presentes na rede com os novos dados, (ii) Congelar algumas camadas e assim limitar o número de parâmetros treinados pela rede, (iii) Adicionar mais camadas no modelo e realizar o treinamento somente destas novas camadas.

A escolha da abordagem de transferência pode variar de acordo com a similaridade do novo conjunto de dados em relação ao antigo e também a seu tamanho (ARAÚJO et al., 2017).

2.5.2 MobileNet

MobileNet é um modelo de RNC criado para apresentar bons resultados em classificações de imagens e ao mesmo tempo, ser leve, tanto no tamanho em memória, quanto no custo operacional de suas operações, isto para que seja possível realizar sua aplicação em *Smartphones* e aplicações embarcadas de visão computacional. Faz isto através da aplicação de técnicas de convolução profunda separável, onde um filtro é aplicado para cada camada de cor, tornando o modelo mais leve (HOWARD et al., 2017).

2.5.3 PoseNet

PoseNet é um modelo de RNC, criado pelo *Google Creative Lab* com base nos trabalhos Papandreou et al. (2017) e Papandreou et al. (2018) que realiza a identificação de 17 pontos do corpo humano (Figura 11), de um ou vários usuários.

Figura 11 – Pontos identificados pelo PoseNet

Fonte – Adaptado de Oved (2018)

Mesmo fazendo muitas identificações (Figura 11), o PoseNet é um modelo leve, o que permite sua aplicação em diversos contexto, como em aplicações *web* e *mobile* (OVED, 2018).

2.6 Conceitos Tecnológicos

Nesta seção as tecnologias utilizadas durante a desenvolvimento do presente trabalho são expostas.

2.6.1 Linguagens de Programação

O processamento e coleta dos dados foi feito utilizando a linguagem de programação Python (ROSSUM, 1995), junto as bibliotecas OpenCV e scikit-image para o processamento das imagens, Augmentor, para a aplicação de modificações nas imagens no processo de *Data Augmentation* e PyQt para a criação de interfaces gráficas.

A criação e treinamento dos modelos de RNCs aplicados nos RA utilizou Python junto a biblioteca Keras, que permite a criação em alto nível de RNAs profundas, além de disponibilizar modelos já treinados e funcionalidades para a aplicação de transferência de aprendizado (CHOLLET, 2015). A disponibilização dos resultados do Keras é feita através

de arquivos hierárquicos (H5), que podem ser consumidos por outras bibliotecas, como as presentes no ecossistema do TensorFlow.

Para o desenvolvimento da biblioteca de RAs, fez-se a utilização da linguagem de programação JavaScript, junto as bibliotecas, P5.js, para uma manipulação facilitada de DOM e TensorFlow.js (TFJS), para o desenvolvimento de Aprendizado Profundo na *web*, isso além de permite a utilização de modelos de rede neural em navegadores, e também disponibiliza modelos já treinados (SMILKOV et al., 2019), da mesma forma que o Keras.

Por fim, para a distribuição do modelo de rede neural treinado neste trabalho, fez-se uma *API Rest* utilizando o *Microframework web* de Python Flask, que permite a construção simplificada de aplicações *web*.

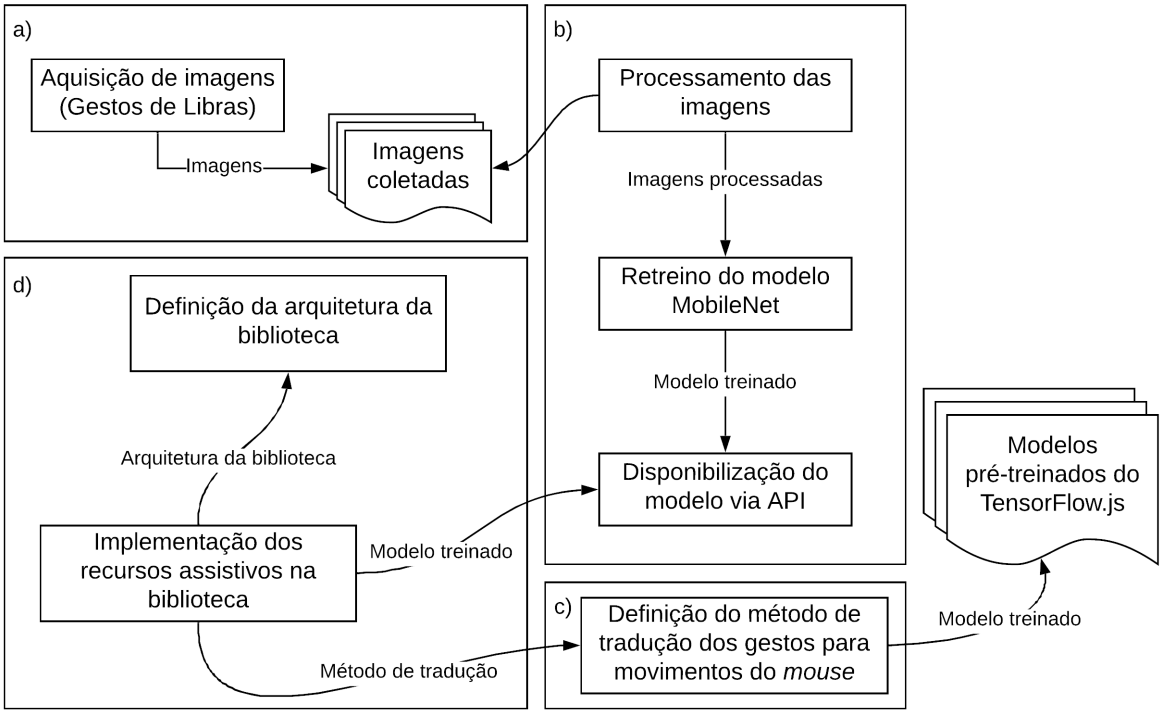
2.6.2 Google Colaboratory

Colaboratory (Colab) é uma ferramenta criada pelo Google que permite a fácil execução de algoritmos de aprendizado de máquina. Colab é criado sob o pacote Jupyter, um ambiente interativo, executado no navegador, que permite a execução de linguagens interpretadas (PÉREZ; GRANGER, 2007). Todo o ambiente do Colab é executado sobre máquinas aceleradas por *Graphics Processing Unit* (GPU), o que diminui o tempo de execução de processos de treinamento de modelos de rede neural.

3 Desenvolvimento

Este capítulo aborda cada uma das etapas do processo de desenvolvimento dos recursos assistivos e implementação da biblioteca ICan.js[1]. Os processos detalhados neste capítulo estão na Figura 12 representados de maneira geral.

Figura 12 – Fluxo de desenvolvimento do projeto



Fonte – Produção do autor

Na Figura 12, inicialmente fez-se a aquisição das imagens de Libras (a), em seguida foi feito o processamento das imagens, treino e distribuição do modelo (b), após isto, a definição do método de conversão da movimentação do usuário para páginas *web* (c) foi definida, por fim, definiu-se a arquitetura da biblioteca e realizou-se a implementação dos recursos assistivos (d).

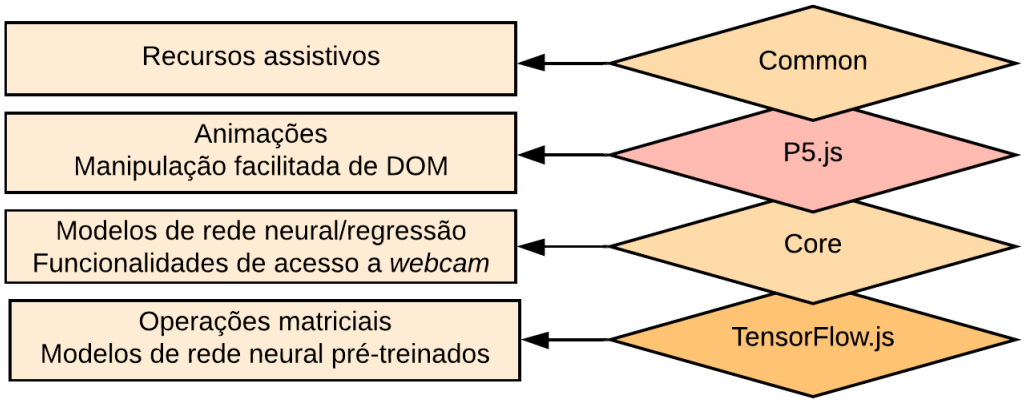
[1] Disponível em: <https://icanjs.netlify.com/>

3.1 Arquitetura da biblioteca

A arquitetura da biblioteca, como apresentado anteriormente, foi desenvolvida após as etapas de processamento e criação dos modelos, porém, para a boa compreensão do desenvolvimento deste trabalho, inicialmente é feito a apresentação da biblioteca e sua arquitetura, o que possibilita a fácil ligação de cada um de seus módulos e funções com as citadas na Figura 12.

A arquitetura do ICan.js, foi criada seguindo a estrutura apresentada por Smilkov et al. (2019), desta forma, a arquitetura é separada em conjuntos de funcionalidades, o que permite um desenvolvimento organizado e uma utilização facilitada. Neste caso, os conjuntos são nomeados `Core` e `Common`, que são construídos sobre as funcionalidades disponibilizadas pelo TensorFlow.js e P5.js respectivamente, como apresentado na Figura 13.

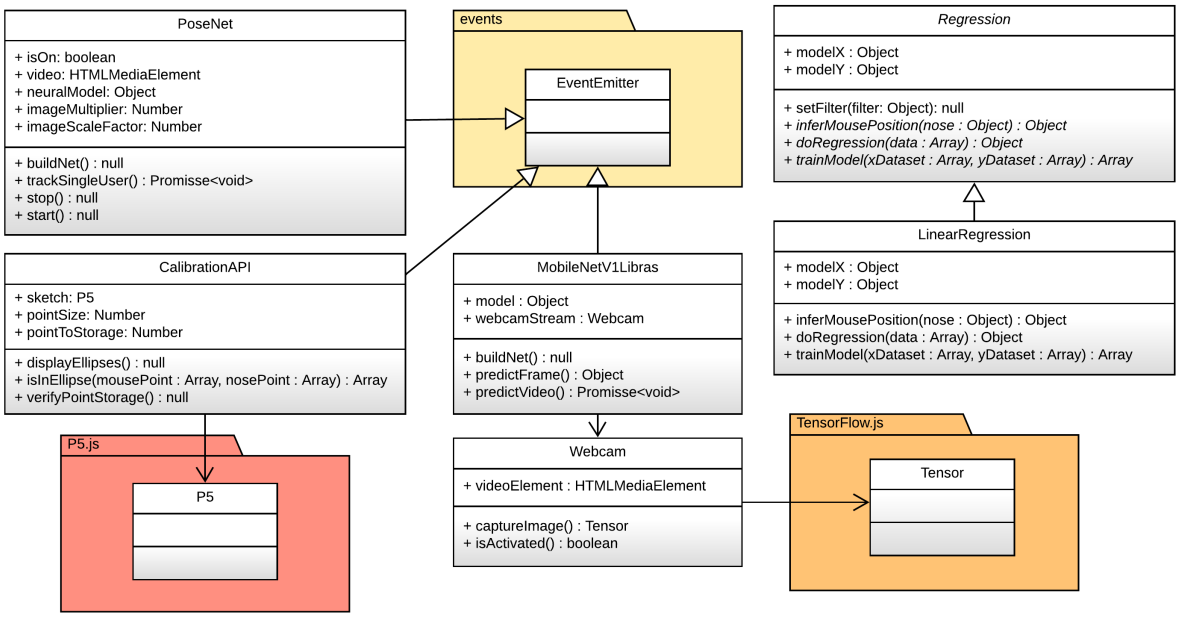
Figura 13 – Arquitetura do ICan.js



Fonte – Produção do autor

O conjunto `Core` é responsável por disponibilizar funcionalidades base para o desenvolvimento dos recursos assistivos, como o acesso a *webcam* dos usuários, os modelos de regressão e também os modelos de rede neural, podendo estes serem carregados de uma *API Rest* de distribuição, ou mesmo do próprio TFJS. Esta camada foi implementada utilizando a especificação ECMAScript 6, o que permitiu o desenvolvimento orientado a objetos mais organizado. As classes desta camada são apresentadas na Figura 14, onde é evidenciado que, as classes `PoseNet`, `CalibrationAPI` e `MobileNetV1Libras` estendem `EventEmmiter` do pacote `events`, isto para que, a interação com estas classes possa ser feita através de eventos, possibilitando a execução assíncrona das classes.

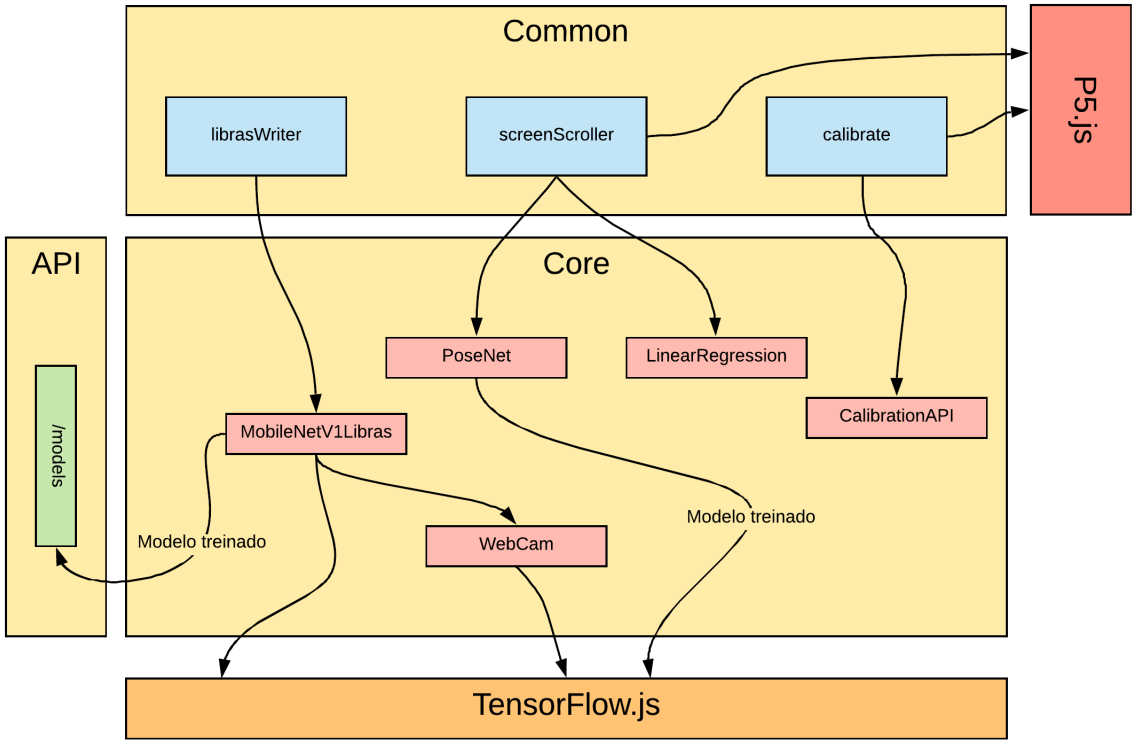
Figura 14 – Diagrama de classes da camada `Core`



Fonte – Produção do autor

Por outro lado, há a camada `Common`, que disponibiliza os recursos assistivos através de funções, criadas com a utilização das funcionalidades da camada `Core` e também da biblioteca P5.js. Na Figura 15 as relações das camadas e bibliotecas utilizadas no ICan.js podem ser vistas.

Figura 15 – Relação entre cada uma das camadas



Fonte – Produção do autor

Para a exposição de cada um dos componentes da Figura 15, as seções seguintes apresentam as etapas do desenvolvimento de cada um dos RAs e seus componentes.

3.2 Tradução de Libras para Texto

Este RA permite a interação dos usuários com deficiência auditiva a páginas *web* através de gestos de Libras. As etapas descritas nas subseções abaixo representam os passos a, b e d da Figura 12.

3.2.1 Aquisição dos dados

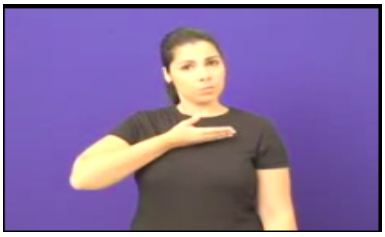
O grande desafio para o desenvolvimento desse RA foi a base de dados, já que, atualmente não há uma base de dados de Libras disponível publicamente (MAGALHãES, 2018), de modo a ser necessário a criação para o presente trabalho.

A seleção dos gestos que fazem a composição da base de dados foi feita seguindo um critério de identificação, neste é necessário que o gesto possa ser identificado com

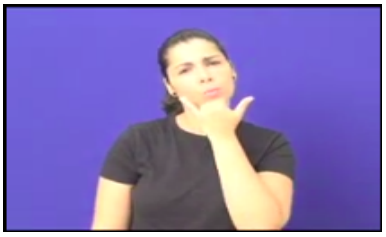
apenas um *frame*, ou seja, apenas uma imagem é o suficiente para a identificação do gesto, o que torna a RNA aplicada mais simples quando comparada a uma que leva em consideração múltiplas imagens. Desta forma os gestos selecionados foram identificados com essa característica por Magalhães (2018), sendo eles (a) Amigo, (b) Desculpa, (c) Telefone, estes apresentados na Figura 16.

Figura 16 – Gestos selecionados para o conjunto de dados

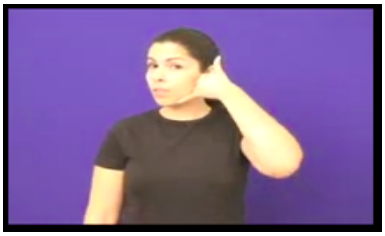
(a) Amigo



(b) Desculpa



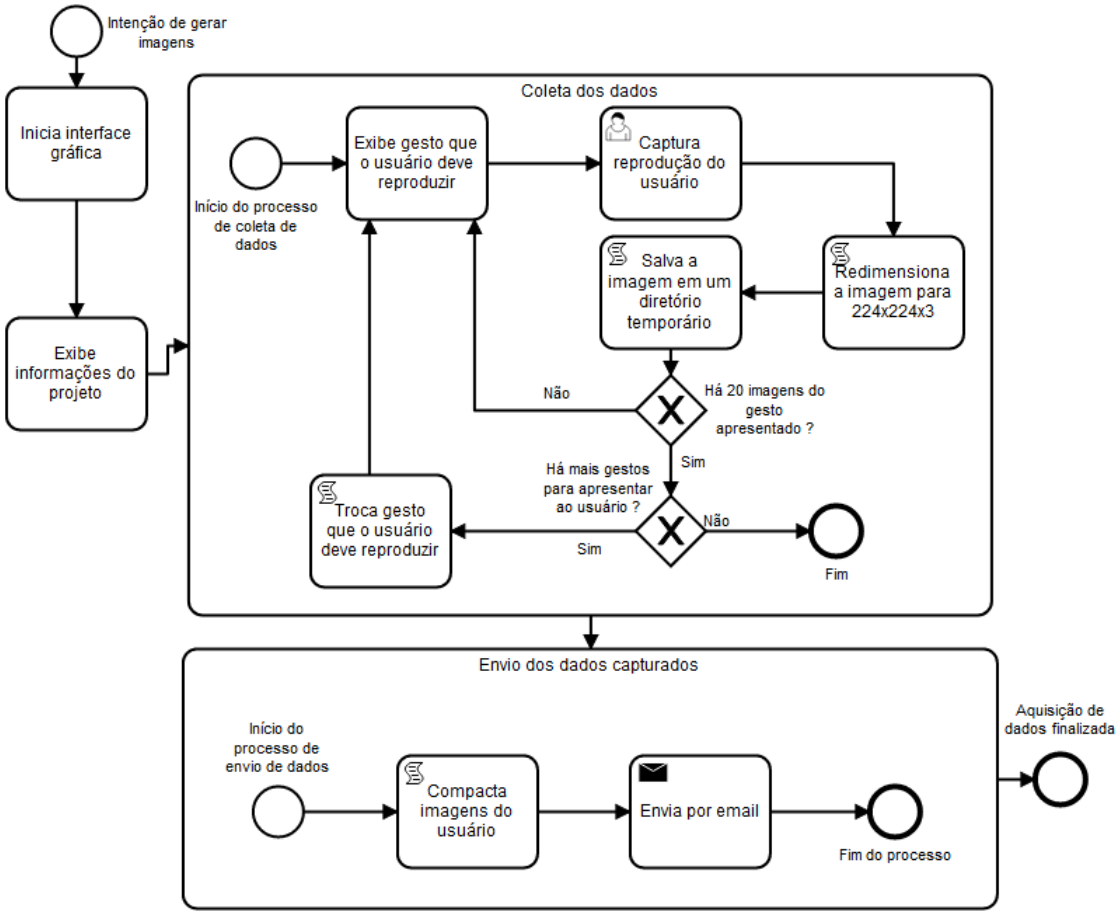
(c) Telefone



Fonte – Adaptado de Dicionário... (2011)

Para a aquisição dos dados, foi desenvolvido uma ferramenta *desktop* multiplataforma na linguagem Python, com a utilização das bibliotecas PyQt e OpenCV. O fluxo de funcionamento da ferramente é apresentado na Figura 17, onde o sistema ao ser iniciado, exibe informações gerais sobre o projeto para o usuário e então já começa o processo de coleta de dados, neste, exemplos dos gestos a serem reproduzidos pelo usuário são exibidos, juntamente a imagem do próprio usuário, além de uma barra de porcentagem para indicar ao usuário seu processo geral na aquisição das imagens.

Figura 17 – Tela inicial da aplicação de aquisição de dados



Fonte – Produção do Autor

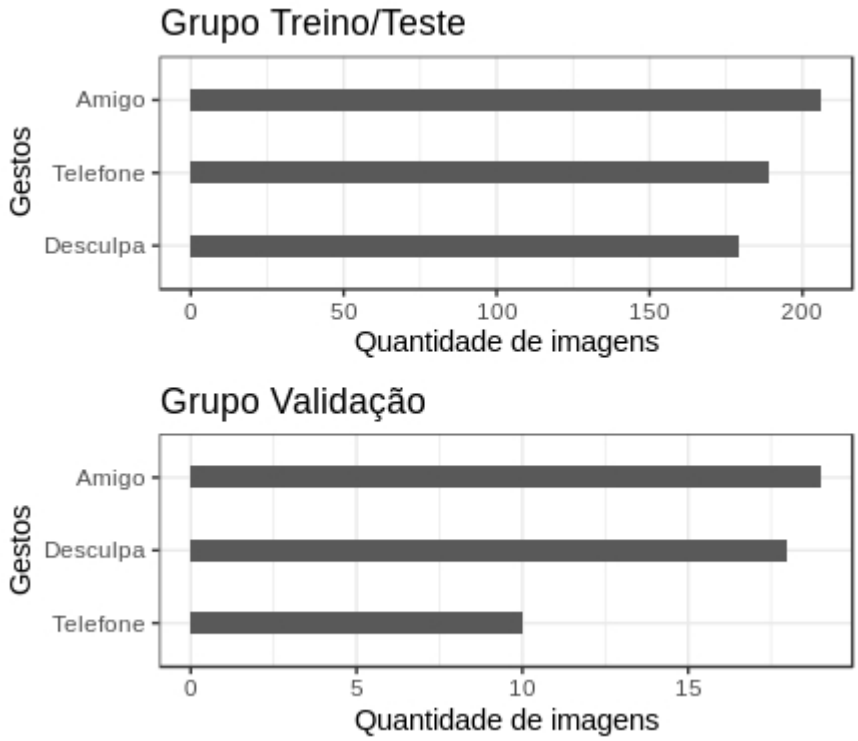
No total, o sistema captura 60 imagens, sendo 20 para cada um dos gestos. Em cada imagem capturada é aplicado uma operação de redimensionamento para que todas as imagens ao final do processo tenham a dimensão 224x224x3, exigida pela RNC utilizada. Após a aquisição o programa compacta as imagens e às envia por *e-mail*.

O programa foi distribuído e ao final houveram 12 colaboradores, criando um conjunto com 720 imagens. Este conjunto de dados foi separado em duas partes, a primeira, com imagens de 11 colaboradores, para o treino e teste do modelo de rede neural, e a segunda, com o colaborador restante, para a validação do modelo. Isso para que, um processo de validação com dados nunca apresentados para a RNC fosse realizado, como forma de assegurar a generalização do modelo, uma vez que, nos dados de treino e teste, mesmo sendo separados nesses dois tipos, possuem imagens dos mesmos participantes.

3.2.2 Pré-processamento dos dados

Durante a aquisição das imagens, nenhuma restrição foi imposta aos colaboradores, para tornar o identificador de imagens o mais geral possível, assim, uma etapa de validação de cada uma das imagens teve de ser realizada para garantir que, cada imagem representa o gesto ao qual está vinculada, o que resultou na remoção de algumas imagens. Este processo foi realizado nos dois grupos de dados criados. A Figura 18 apresenta a relação da quantidade de imagens com cada um dos gestos após a verificação dos dados.

Figura 18 – Quantidade de imagens por gesto após verificação em cada grupo



Fonte – Produção do Autor

Após a filtragem, o grupo de dados para treino e teste foi dividido, ficando 80% dos dados para treino e o restante para teste. Este processo foi criado utilizando a biblioteca sklearn, e o código é apresentado na Figura 19.

Figura 19 – Script de separação dos dados (Treino X Teste)

```
1  from sklearn.model_selection import train_test_split
2
3  x, y = [], []
4
5  for collaborator_dir in os.listdir():
6      files = os.listdir(collaborator_dir)
7
8      x.extend(files)
9      y.extend(np.repeat(collaborator_dir, len(files)))
10
11 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
       =0.20, random_state=992)
```

Fonte – Produção do autor

Nas linhas 5 a 9, a referência do arquivo de cada um dos colaboradores é colocada dentro de listas, cada uma delas representando respectivamente, o arquivo e a classe a qual o arquivo representa, após isto, na linha 11, as listas são divididas em treino e teste.

Com a divisão realizada, cada uma das imagens divididas em treino e teste são movidas para os diretórios de seus respectivos tipos com uma função criada em Python (Figura 20), esta função recebe uma lista com as referências dos arquivos e suas respectivas classes e então copia os arquivos para os devidos diretórios.

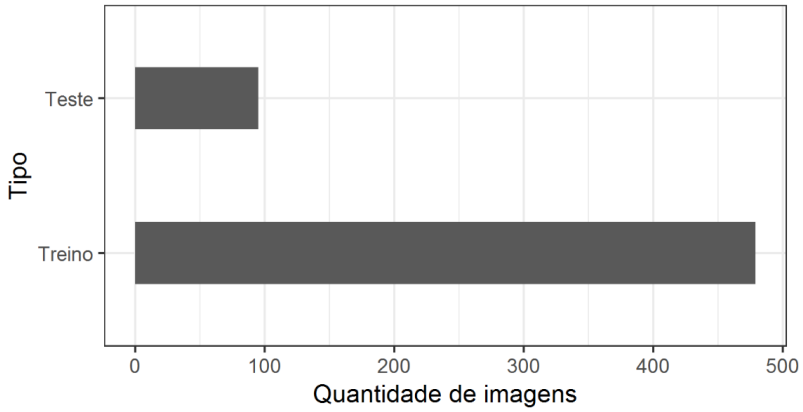
Figura 20 – Função para movimentação dos arquivos de Treino e Teste

```
1  def move_data(x_data: list, y_data: list, typeof: str) -> None:
2      for xt, yt in zip(x_data, y_data):
3          src = os.path.join(yt, xt)
4          dst = os.path.join(typeof, yt, xt)
5
6          copyfile(src, dst)
```

Fonte – Produção do autor

Com a finalização da separação dos dados, as quantidades de imagens nos conjuntos de treino/teste podem ser vistas na Figura 21.

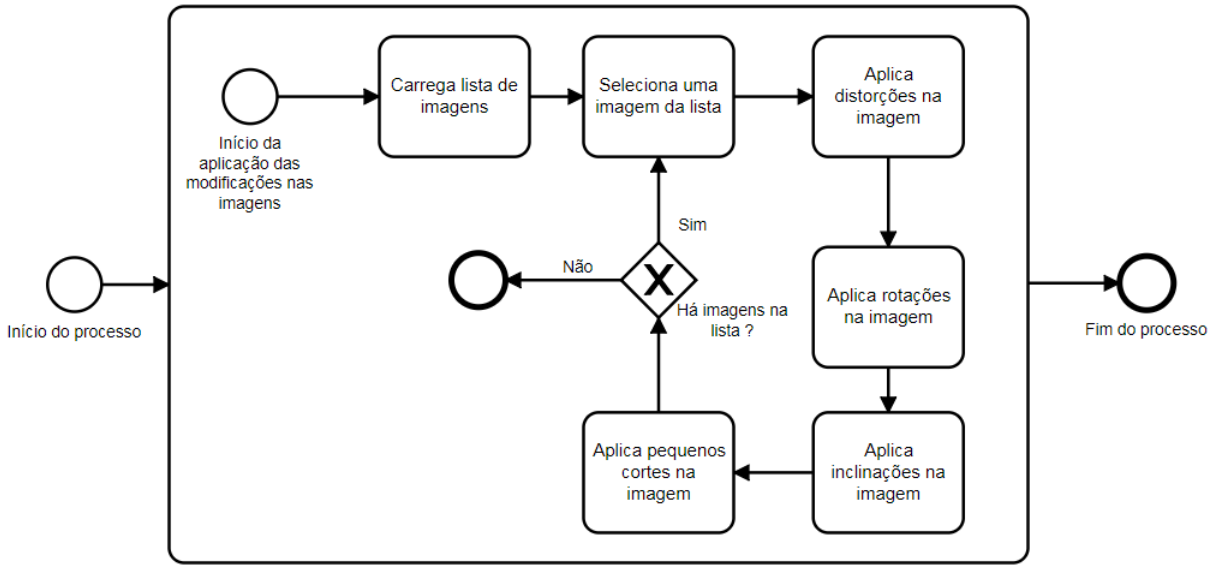
Figura 21 – Quantidade de imagens de treino e teste



Fonte – Produção do Autor

Poucas imagens podem ser um problema para a generalização do modelo, mesmo levando em consideração a transferência de aprendizado do MobileNet que será feita, assim é aplicado no conjunto de treino, separado anteriormente o *Data Augmentation*. A utilização desta técnica foi feita através da biblioteca Augmentor, onde em um processo iterativo, que passa por todas as imagens do conjunto aplicando certas alterações, e cada uma destas alterações possuem uma probabilidade de ocorrência. A Figura 22 apresenta o fluxo de processamento do Augmentor e todas as alterações que foram configuradas neste trabalho.

Figura 22 – Fluxo de alterações do processo de *Data Augmentation*



Fonte – Produção do Autor

A implementação do fluxo da Figura 22 é feita no trecho de código na Figura 23, nesta, na linha 3 cria-se a instância de um processo de alterações, referenciando o diretório onde estão as imagens que são modificadas para a geração das novas imagens. Da linha 5 a 8 são feitas definições de algumas modificações a serem aplicadas nas imagens, e por fim, na linha 10, 700 imagens são geradas através deste processo.

Figura 23 – Trecho do *Script* de *Data Augmentation*

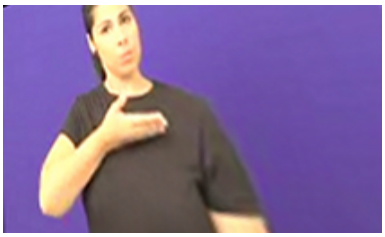
```
1  import Augmentor
2
3  p = Augmentor.Pipeline('gestos_editados/treino')
4
5  p.random_distortion(probability=0.5, grid_height=3, grid_width=3,
       magnitude=2)
6
7  p.skew_left_right(probability=0.3, magnitude=0.7)
8  p.skew_corner(probability=0.4, magnitude=0.5)
9
10 p.sample(700)
```

Fonte – Produção do autor

A Figura 24 mostra exemplos de imagens geradas com as diferentes modificações indicadas no código.

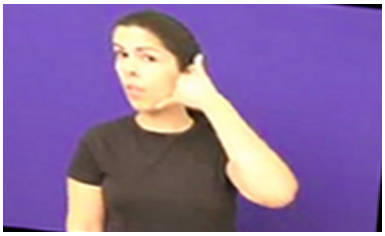
Figura 24 – Exemplos de imagens geradas pelo processo de modificação

(a)



(b)

(c)

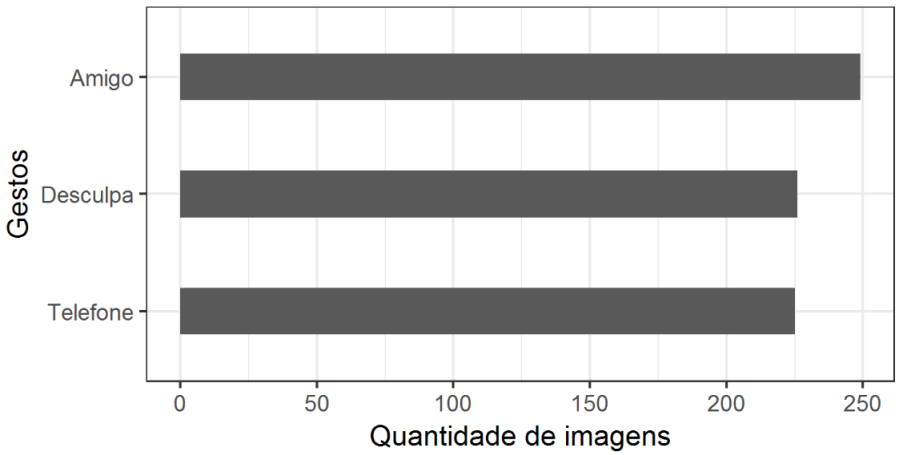


Fonte – Produção do autor

Após a aplicação do *Data Augmentation*, as 700 imagens geradas foram utilizadas como o conjunto de imagens de treino, isto porque, mesmo não havendo uma padrão no momento da aquisição da imagem, muitas delas acabaram ficando semelhantes, por conta

da forma de aquisição. A Figura 25 mostra este valor distribuído por gesto. Com isto, o conjunto de imagens está pronto para ser utilizado no processo de treinamento e teste do modelo de RNC.

Figura 25 – Quantidade de imagens por gesto com *Data Augmentation*



Fonte – Produção do Autor

3.2.3 Treinamento da Rede Neural Convolucional

O modelo de RNC utilizado neste RA foi o MobileNet, carregado da biblioteca Keras já treinado no conjunto de dados ImageNet (RUSSAKOVSKY et al., 2015). O modelo carregado foi empregado no desenvolvimento da RA através da técnica de transferência de aprendizado.

Esse processo começa com a criação da instância de MobileNet já treinada. Nessa instância não existem a camada de classificação, esta especifica para cada problema, não sendo reutilizável, assim essa camada é adicionada ao modelo instanciado. Na Figura 26 o trecho de código para essa criação é apresentado, nesse, na linha 2 é criada a instância do MobileNet, nas linhas 5 e 6 se cria a camada de saída, com um filtro, por fim, nas linhas 9 e 12, são criados o classificador e o novo modelo configurado.

Figura 26 – Trecho de código para a criação do modelo

```
1  # Criando o modelo que será retreinado
2  mobile_net = MobileNet(input_shape=(224, 224, 3), weights='imagenet',
       include_top=False)
3
4  # Configurando o novo topo da rede neural
5  model_top = mobile_net.output
6  model_top = GlobalAveragePooling2D()(model_top)
7
8  # Classificador
9  model_top_classifier = Dense(3, activation='softmax')(model_top)
10
11 # Gerando novo modelo
12 new_mobile_net = Model(inputs=mobile_net.input, outputs=
       model_top_classifier)
```

Fonte – Produção do autor

Como citado, neste RA aplicou-se a técnica de transferência de aprendizado no MobileNet, assim foram selecionados para o treinamento todas as camadas após o segundo bloco de convolução, aproveitando as especifidades dos blocos até o segundo e buscando novas características nas camadas a frente. Desta maneira, na Figura 27 está o trecho de código para criação do modelo. Na linha 1 todos os blocos de convolução após o segundo são postos para treinamento.

Figura 27 – Trecho de código da seleção dos blocos a serem treinados

```
1  new_mobile_net = unfreeze_layers_from(new_mobile_net, 20)
```

Fonte – Produção do autor

Com a definição da camada de classificação e das camadas que serão retreinadas, o modelo é configurado com as métricas, funções de otimização e perda utilizadas durante o treinamento. A configuração é feita com o código da Figura 28.

Figura 28 – Trecho de código da seleção dos blocos a serem treinados

```
1  new_mobile_net.compile(optimizer=Adam(lr=1e-3), \
2          loss='categorical_crossentropy', metrics=['accuracy'])
```

Fonte – Produção do autor

Após as configurações, o treinamento do modelo foi realizado com 5 épocas, onde cada uma das épocas representa a quantidade de vezes em que todas imagens do conjunto de dados foi apresentada para a RNC. Todo o processo de treinamento foi realizado utilizando a plataforma Colab com o uso de GPUs, melhorando o resultado e diminuindo

o tempo de processamento. A Figura 29 mostra o trecho de código da inicialização do treinamento.

Figura 29 – Trecho de código do treinamento do modelo

```
1 new_mobile_net.fit_generator(train_generator,
2                               steps_per_epoch=8,
3                               epochs=5,
4                               validation_data=test_generator,
5                               validation_steps=2)
6
7 new_mobile_net.save('results/icanv2.h5')
```

Fonte – Produção do autor

No trecho de código da Figura 29, na linha 7, é evidenciado que o modelo será salvo no formato H5, porém a utilização desse modelo no TFJS requer uma conversão para um formato JSON. Esta conversão é feita na Figura 30 e utiliza a ferramenta `tensorflowjs_converter`.

Figura 30 – Conversão de formato do modelo treinado

```
1 tensorflowjs_converter --input_format keras results/icanv2.h5 \
2                                    results_tfjs/
```

Fonte – Produção do autor

A utilização da ferramenta (Figura 30) recebe o modelo salvo no formato H5 e o diretório onde o modelo no formato JSON será salvo.

3.2.4 Distribuição do modelo

Com a finalização do treinamento e conversão do modelo, foi necessário criar uma forma que facilitasse sua distribuição, uma vez que, para a utilização o TFJS carrega o modelo e todos os seus pesos. Neste caso foi optado pela criação de uma *API Rest*[2] para a distribuição do modelo, o que evita aos usuários do ICan.js qualquer necessidade de criar suas próprias formas de distribuição do modelo.

A *API Rest* foi criada utilizando Python junto ao Flask. A Figura 31 apresenta o código da rota criada para a distribuição do modelo de reconhecimento de Libras.

[2] Disponível em: <https://ican-api.herokuapp.com/>

Figura 31 – Rota de distribuição do modelo de reconhecimento de Libras

```
@app.route("/models/mobilenetv1/<file>", methods=["GET"])

@cross_origin()
def mobilenetv1_model(file):
    path = os.path.join(app.config["BASE_DIR"], \
                        "api/models/mobilenetv1")

    try:
        return send_from_directory(directory=path, filename=file)
    except:
        return jsonify({
            "error": True,
            "message": "Erro ao tentar recuperar os dados"
        })
```

Fonte – Produção do autor

Para consumir esta *API* e carregar o modelo na biblioteca, criou-se a classe `MobileNetV1Libras`, que está no conjunto `Core` de funcionalidades do ICan.js. O código de consumo da *API* é mostrado na Figura 32.

Figura 32 – Método de consumo da *API REST* com o modelo de reconhecimento de Libras

```
async buildNet() {
    if (this.model === null) {
        this.model = await tf.loadModel(new URL("/models/mobilenetv1/
            model.json", MODEL_URL).href);
    }
}
```

Fonte – Produção do autor

Para que o uso da biblioteca seja rápido, este modelo é carregado uma vez em cada instância de `MobileNetV1Libras`, desta forma, na linha 2 da Figura 32 é verificado se o modelo já foi carregado, caso não tenha sido carregado, a linha 3 é executada e através do TFJS o modelo é carregado em memória.

3.2.5 Criação do recurso assistivo

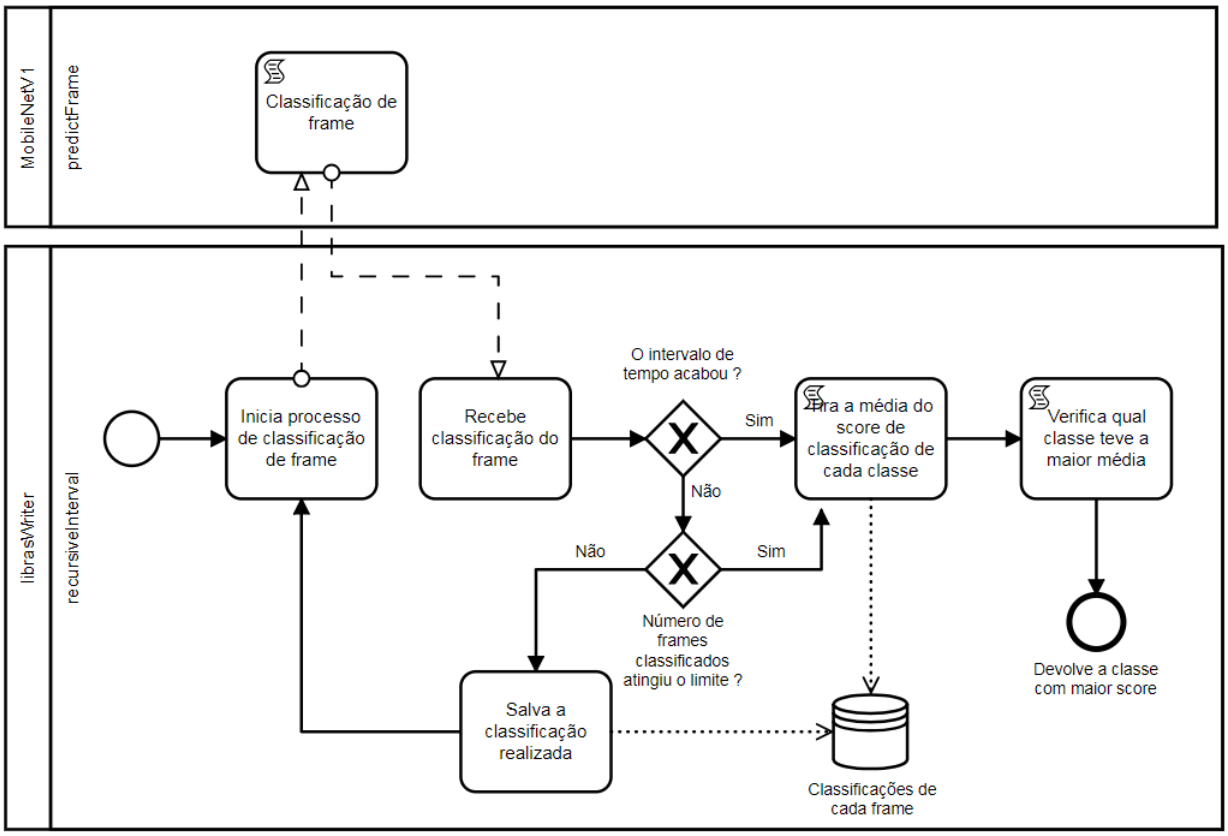
Após todo o processo de coleta de dados, treinamento, disponibilização e consumo do modelo da RNC utilizada neste RA, foi adicionado na camada `Common` a função `librasWriter`, que permite a escrita em campos de páginas *web* utilizando gestos de Libras.

Como mostrado anteriormente, as imagens utilizadas durante o treinamento do modelo são estáticas, o que faz o `librasWriter` transcrever a classificação de uma única

imagem para texto, porém para garantir a usabilidade, a função também realiza a classificação contínua, isto através do método descrito por Magalhães (2018), onde a média dos resultados de classificação feitas em cada uma das imagens capturadas em um intervalo de tempo é utilizado para definir qual foi o gesto realizado pelo usuário. Desta forma, mesmo que o modelo MobileNet retreinado neste trabalho utilize apenas imagens estáticas, pode-se ter uma identificação contínua dos gestos feitos pelo usuário.

Essa implementação foi feita no ICan.js permitindo ao usuário inserir o intervalo de tempo (em segundos) que deve ser considerado e também a quantidade de imagens capturadas. O processo de funcionamento da implementação é detalhado na Figura 33.

Figura 33 – Processo de classificação de um conjunto de *frames*



Fonte – Produção do Autor

Para a implementação do processo da Figura 33 no ICan.js, foi adicionado uma função recursiva (Figura 34) dentro do `librasWriter`, onde, a linha 3 realiza o processo de classificação de uma imagem, na linha 5 ocorre a verificação da quantidade de imagens já capturadas, e caso a quantidade seja maior ou igual ao definido pelo usuário, a média das classificações é calculada para definir o sinal que em média mais apareceu nas imagens capturadas e então o resultado é enviado para o usuário em um *callback*. Nas linhas 10 a

12 é executado o processo de espera para que, um novo intervalo de tempo seja iniciado.

Figura 34 – Função de classificação recursiva

```
async function recursiveInterval() {
    try {
        gestures.push(await mobilenetGestures.predictFrame());

        if (gestures.length >= nFrames) {
            fnc(getMeanGesture(gestures));
            gestures = [];
        }

        timeout = window.setTimeout(() => {
            recursiveInterval();
        }, delay * 1000);
    } catch(err) {
        if (timeout !== null) {
            window.clearTimeout(timeout);
        }

        console.error("librasWriter", err);
    }
}
```

Fonte – Produção do autor

Desta forma, os resultados vindos desta função podem ser inseridos em qualquer campo ou formulário de uma página, dependendo apenas da forma como é utilizada no *site*.

3.3 Controle do *cursor* do *mouse* com movimentos da cabeça

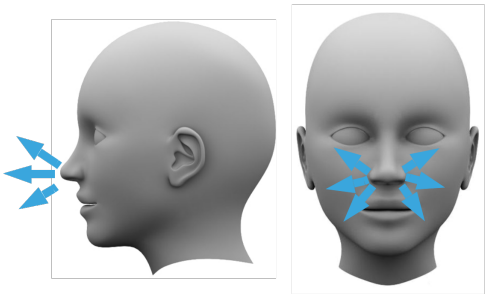
Este recurso assistivo permite a interação de usuários com deficiência motora a páginas *web* através da movimentação da cabeça. As etapas descritas nas seções abaixo são apresentadas na Figura 12 (c, d).

3.3.1 Definição do ponto de referência

Para iniciar a criação deste recurso assistivo, foi necessário antes realizar a definição do que seria utilizado como referência para identificar as ações do usuário e mapeá-las para movimentos no *cursor* do *mouse* nas páginas *web*.

Este mapeamento pode ser feita de diferentes formas, com técnicas variadas como as apresentadas em Griffin e Ramirez (2018) e Papoutsaki e Laskey (2016). No contexto deste trabalho decidiu-se utilizar o nariz, que servindo como um ponto de referência, permite ao usuário apontar para qualquer parte da tela (Figura 35).

Figura 35 – Movimentação do usuário com o nariz



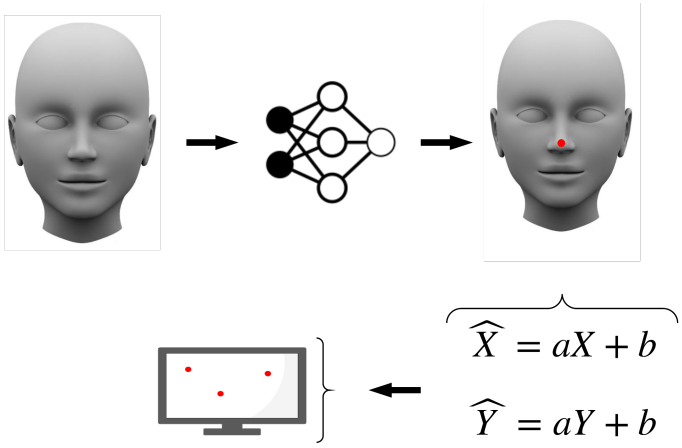
Fonte – Produção do Autor

Com esta definição realizada faz-se necessário aplicar técnicas para a identificação e utilização deste ponto de referência.

3.3.2 Mapeamento dos movimentos do usuário

O mapeamento dos movimentos do usuário, considerando o nariz, exigiu a criação de um processo separado em duas partes, a primeira parte, responsável por identificar o ponto de referência na imagem do usuário e a segunda parte, que através dos resultados de identificação gera os movimentos do *cursor* do *mouse*. Para isso, na etapa de identificação decidiu-se utilizar o modelo PoseNet, e então, o resultado da identificação é aplicado em regressões lineares e seus resultados utilizados para gerar os movimentos do *cursor*, como resumido na Figura 36.

Figura 36 – Fluxo de funcionamento do mapeamento de movimentos para tela do computador



Fonte – Produção do Autor

A aplicação da primeira parte do fluxo (Figura 36) no ICan.js é feita através da classe `PoseNet` (Figura 15), presente na camada de funcionalidades `Core` da biblioteca. Esta classe carrega o modelo PoseNet (Figura 37) já implementado no TFJS e disponibiliza resultados fazendo o consumo deste modelo.

Figura 37 – Método de carregamento do modelo PoseNet

```
1 async buildNet() {
2     if (this.neuralModel === null) {
3         this.neuralModel = await posenet.load(this.imageMultiplier);
4     }
5 }
```

Fonte – Produção do autor

A forma de carregar o modelo do TFJS (Figura 37) não é muito diferente ao apresentado na Figura 32, que carrega o modelo da *API Rest*, a diferença é que, na linha 3, o módulo `posenet` do TFJS é utilizado para carregar o modelo.

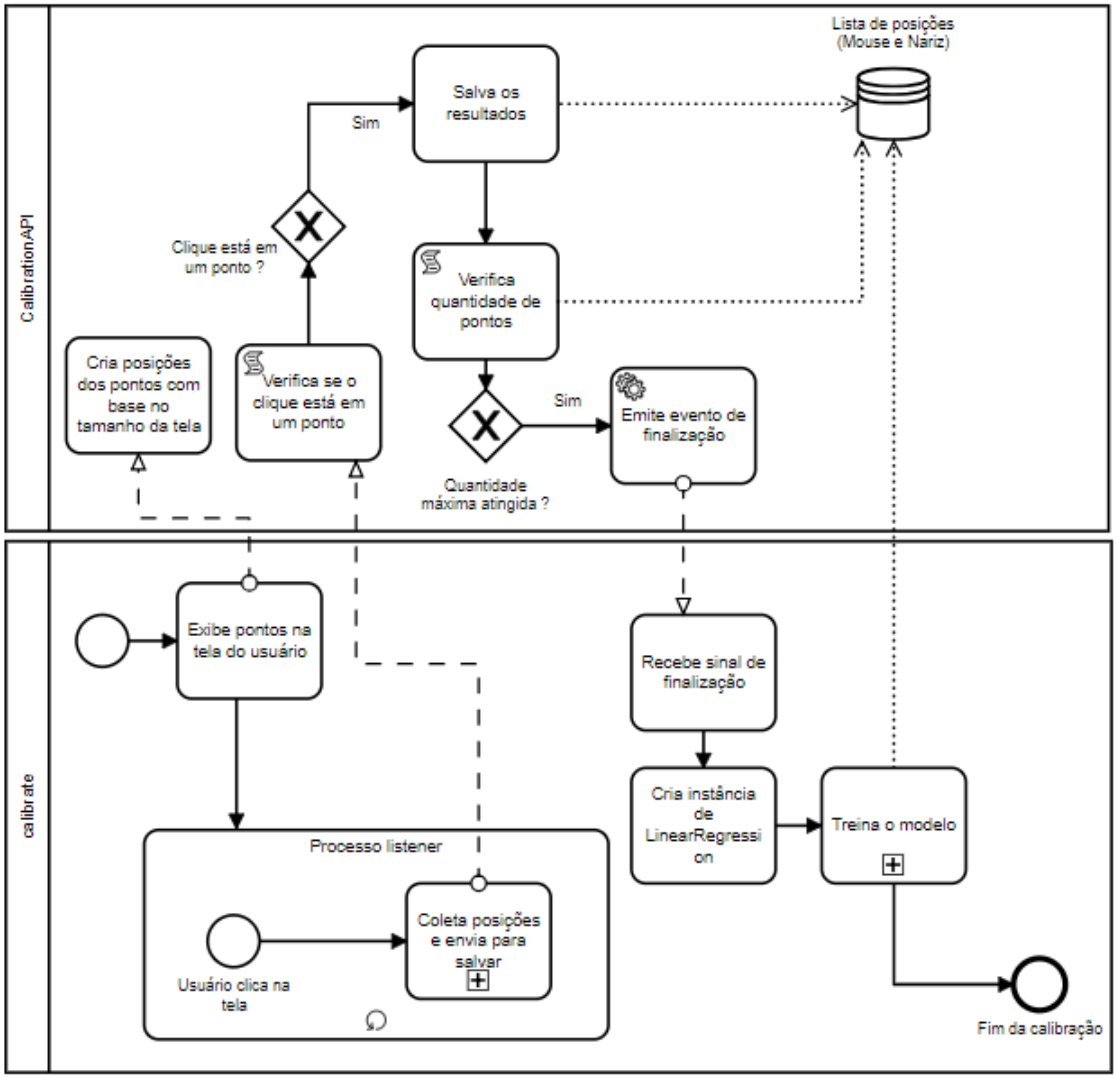
Com a identificação da posição do nariz do usuário sendo feita, foi iniciada a segunda etapa do processo de mapeamento dos movimentos, como citado, utilizando regressões lineares. A forma de aplicação das regressões lineares utilizadas neste trabalho no contexto de mapeamento de movimentos, foram descritas por Papoutsaki e Laskey (2016), onde para que a regressão pudesse ser aplicada neste contexto dois modelos são criados, um para a estimativa de posições horizontais, que possuem como variáveis dependentes e independentes, respectivamente, as posições X do *cursor* e do ponto de referência do usuário, neste caso o nariz. Para o segundo modelo, de posições verticais, a mesma lógica é aplicada, porém as variáveis utilizadas são as posições Y, do *cursor* e do nariz do usuário. Esta técnica foi implementada na camada `Core` através da classe `Regression` e suas especializações (Figuras 14 e 15), onde cada uma destas classes possuem os modelos horizontais e verticais respectivamente.

A geração de cada um dos modelos necessita de um processo de ajuste dos coeficiente de regressão, assim no ICan.js foi implementado uma *API* de calibração, que fornece aos utilizadores da biblioteca facilidades para a geração dos coeficientes.

3.3.2.1 Geração dos coeficientes

A geração dos coeficientes de regressão é parte importante para a utilização dos modelos de regressão linear citados anteriormente, para isto, o ICan.js disponibiliza uma *API* de calibração, como apresentado anteriormente. O funcionamento geral desta *API* é evidenciado na Figura 38 e detalhado em seguida.

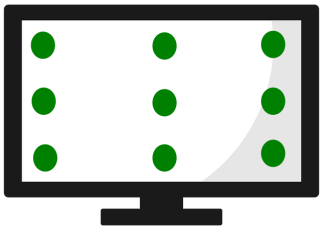
Figura 38 – Fluxo de funcionamento da API de calibração



Fonte – Produção do Autor

Para a coleta dos pontos utilizados na geração dos coeficientes, fez-se uma implementação que também segue as recomendações apresentadas por Papoutsaki e Laskey (2016), nessa uma matriz de pontos que cobre as principais posições da tela, possuindo dimensões 3x3 é exibida na tela do usuário (Figura 39), sendo que, quando o usuário clica em cada um destes pontos é salvo a posição do *cursor* e também do nariz do usuário, capturada pelo PoseNet.

Figura 39 – Tela de Calibração



Fonte – Produção do Autor

Para a implementação da forma de coleta de pontos, inicialmente criou-se no ICan.js a classe `CalibrationAPI` na camada `Core`, que fornece regras das posições dos pontos na tela do usuário, além do controle da quantidade de pontos já registrados. Outro recurso implementando na biblioteca foi a função `calibrate` da camada `Common`, que junto a biblioteca P5.js gera os pontos especificados pela `CalibrationAPI` na tela do usuário, coleta as posições do *cursor* (Figura 40) e também do ponto de referência do usuário, neste caso, do nariz, gerado pelo PoseNet, além de gerar os modelos de regressão linear e seus coeficientes.

Figura 40 – Função de coleta de pontos

```
1  sketch.mousePressed = function() {
2      if (poses !== null) {
3          let noseObj = {
4              x: poses.keypoints[0].position.x,
5              y: poses.keypoints[0].position.y
6          }
7
8          let mouseObj = {
9              x: sketch.mouseX,
10             y: sketch.mouseY
11         }
12         calibrationAPI.isInEllipse(mouseObj, noseObj);
13     }
14 }
```

Fonte – Produção do autor

Na Figura 40 das linhas 3 a 11 os dados de X e Y tanto do *cursor* quando do nariz são postos dentro de objetos que são enviados para uma instância de `CalibrationAPI` na linha 12.

Com a quantidade de pontos armazenados atingindo o exigido pelas regras do

`CalibrationAPI`, um evento é enviado para a função `calibrate`, assim esta cria uma instância da classe `LinearRegression`, uma especialização de `Regression` (Figura 14), separa os pontos coletados em conjuntos contendo respectivamente as posições X e Y e os utiliza para, através do método `trainModel` (Figura 41) da instância de `LinearRegression`, gerar os coeficientes.

Figura 41 – Função de calibração dos modelos

```
1 trainModel(xDataset, yDataset) {
2     // Verificações omitidas
3
4     // Realiza as regressões para cada um dos datasets
5     this.modelX = this.doRegression(xDataset);
6     this.modelY = this.doRegression(yDataset);
7 }
```

Fonte – Produção do autor

O método presente na Figura 41, nas linhas 5 e 6 são executados os métodos de treinamento dos modelos de regressão.

3.3.3 Criação do recurso assistivo

Com a finalização da implementação do PoseNet e da forma de calibração e utilização do modelo de regressão, foi adicionado na camada `Common` a função `screenScroller`, que representa o RA propriamente dito, nesta através da integração das saídas do modelo PoseNet e um modelo de regressão linear já calibrado para o usuário corrente, faz-se as predições do local para onde o usuário está apontando e com o resultado da predição uma `div` representando um *cursor* criada dentro da página pelo ICan.js tem sua posição alterada para o local predito, isto é feito através do método da Figura 42.

Figura 42 – Função de predição

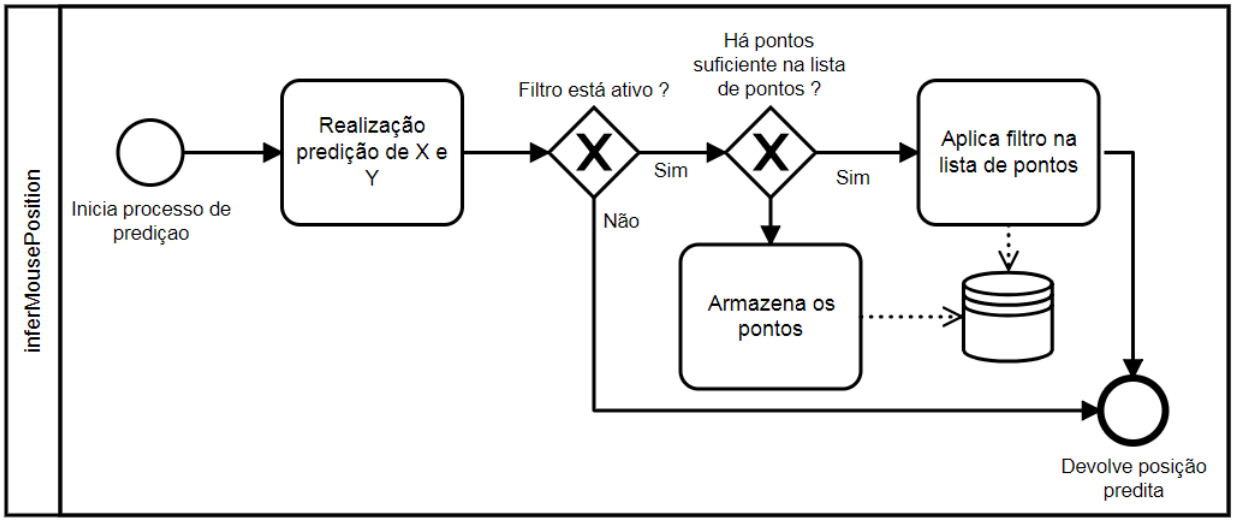
```
1  sketch.draw = function() {
2      if (poses !== null) {
3          let nose = poses.keypoints[0];
4
5          let posObj = regressionModel.inferMousePosition(nose);
6          changeDivPosition(pointer, posObj.x, posObj.y);
7
8          // Lógica de scrolling
9          if (posObj.y < 0) {
10             window.scrollTo(0, window.scrollY + posObj.y * 0.05);
11         } else if (posObj.y > window.innerHeight) {
12             window.scrollTo(0, window.scrollY + (posObj.y - window.
                   innerHeight) * 0.03);
13         }
14     }
15 };
```

Fonte – Produção do autor

Na Figura 42, a linha 1 indica que este método está sendo utilizado dentro de uma instância de `P5`, sendo que, o método `draw` cria um laço infinito, o que faz o método descrito estar sempre sendo executado. Já na linha 3, a posição do nariz é recuperada e então na linha 5 passada para uma instância de `LinearRegression` para que a predição seja realizada, na linha 6 a `div` criada para representar um *cursor* na página é movida para a posição predita e então das linhas 9 à 13, há uma verificação para saber se é necessário subir ou descer a página.

A predição pode representar certa instabilidade, muitas vezes por conta da calibração, desta forma, o ICan.js através da classe `LinearRegression`, fornece a possibilidade da aplicação de filtros de média e mediana em um conjunto com N predições, onde este N é definido pelo usuário, assim ao atingir a quantidade de predições necessárias, aplica-se a média ou a mediana nestas predições e o resultado é devolvido. O funcionamento do fluxo de aplicação destes filtros são apresentados na Figura 43.

Figura 43 – Fluxo de funcionamento dos filtros nos resultados das regressões



Fonte – Produção do Autor

Esta técnica de estabilização das predições permite que a movimentação do *cursor* criado seja mais fluída e assertiva, melhorando a usabilidade.

4 Resultados

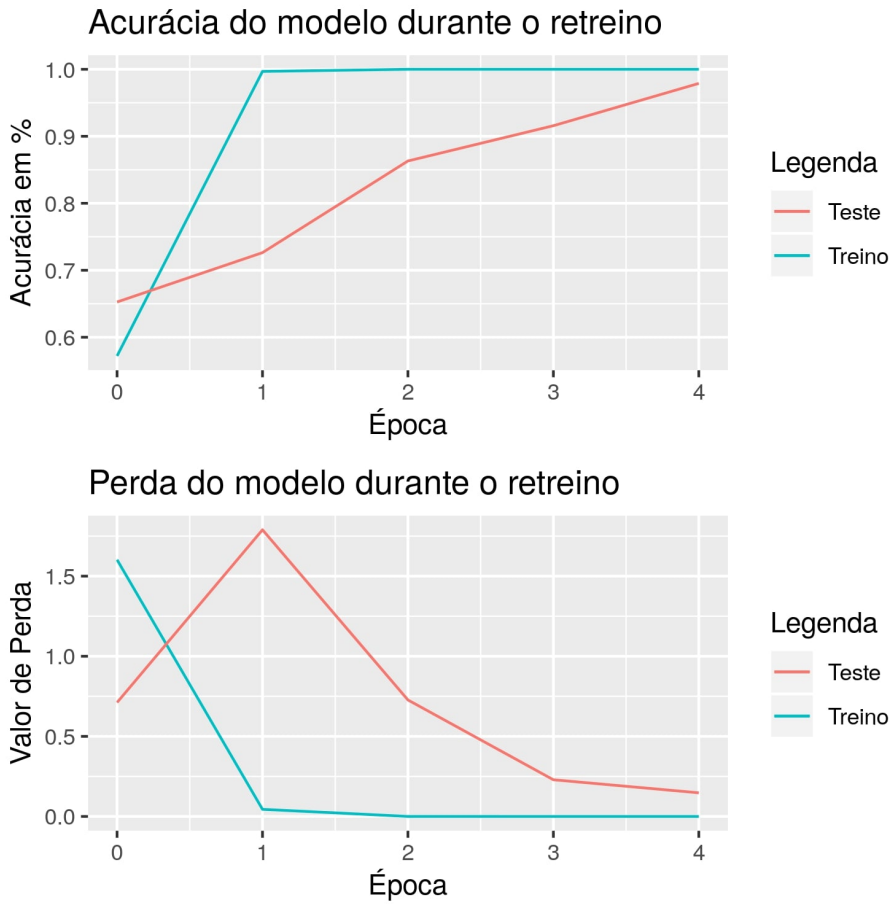
Neste capítulo serão apresentados os resultados do modelo gerado e as páginas *web* criadas para testar os módulos implementados no ICan.js

4.1 Tradução de Libras para texto

Nesta seção, os resultados obtidos com a transferência de aprendizado realizada no modelo MobileNet e também as páginas de testes deste RA são apresentadas.

4.1.1 Resultados da transferência de aprendizado do modelo MobileNet

Figura 44 – Resultado do retreino do MobileNet



Fonte – Produção do Autor

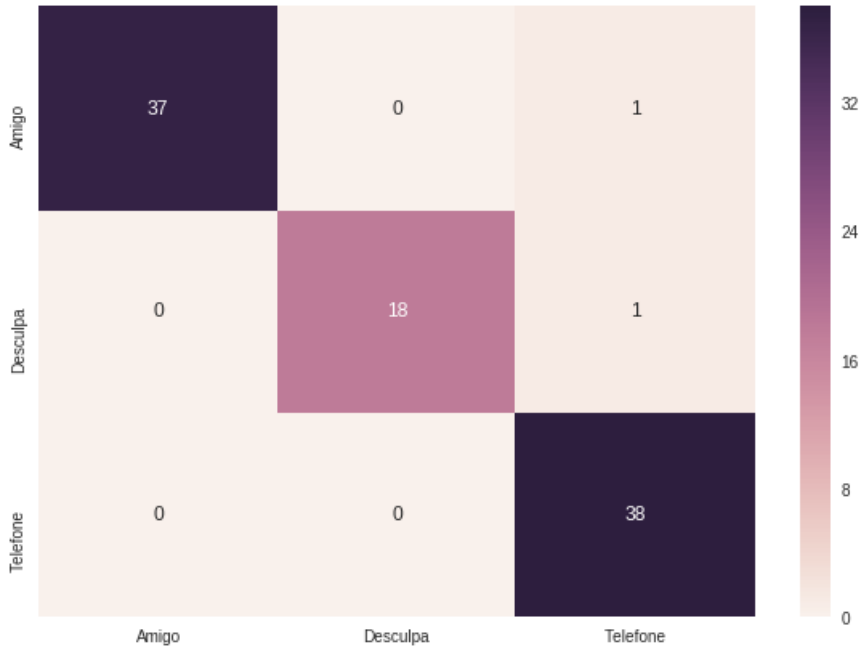
A Figura 44 mostra a acurácia e a perda do modelo durante seu retreino, nota-se que, o processo de retreino levou 5 épocas para atingir bons resultados, isto, tanto na diminuição da perda, quanto no aumento da acurácia do modelo, que nos dois casos, nas primeiras épocas não estavam apresentando bons resultados.

O comportamento dos resultados na Figura 44, onde, mesmo os dados de treino sendo 100% identificados a partir da segunda época e tendo poucas perdas, para os dados de teste, tem-se uma variação inversamente proporcional entre a crescente da acurácia e a diminuição da perda.

Para avaliar o desempenho do modelo, além dos resultados da Figura 44, fez-se a utilização da matriz de confusão, esta que vincula as classes esperadas com a preditas em cada um dos valores presentes nos dados. Como citado anteriormente, os dados foram separados em dois conjuntos, o primeiro para treino/teste e o segundo para validação, isto para garantir que, o modelo gerado de nenhuma forma possui bons resultados somente por conta de um viés aos dados. Desta forma, duas matrizes de confusão foram criadas, uma para o conjunto de treino e teste (Figura 45) e outra para o conjunto de validação (Figura 46).

A matriz de confusão da Figura 45, utilizou apenas os dados de teste, e mesmo nestes havendo imagens de participantes que foram apresentados ao modelo nos dados de treino, uma pequena quantidade de erro foi registrada nas classes Amigo e Desculpa.

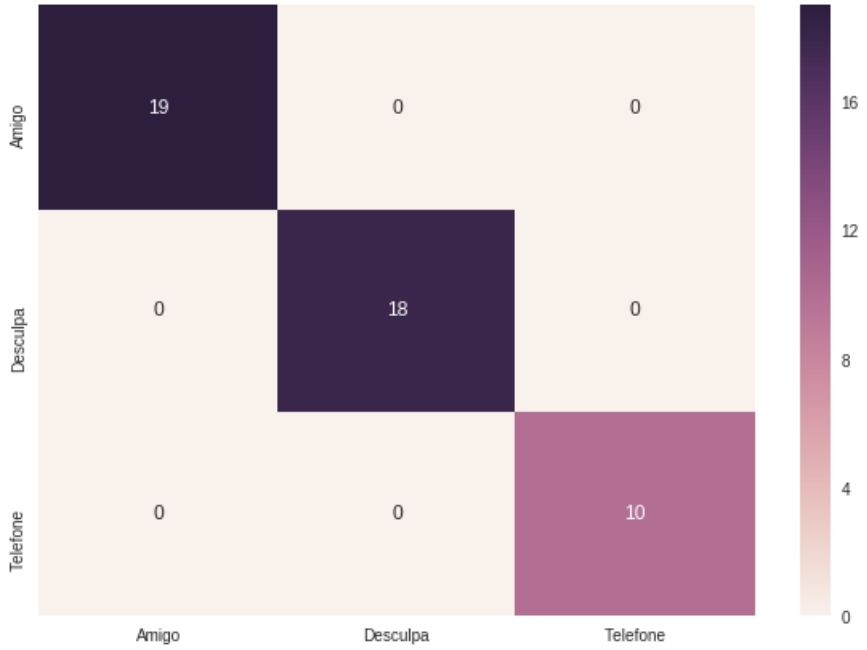
Figura 45 – Matriz de confusão do modelo MobileNet retreinado com dados de teste



Fonte – Produção do Autor

Já na matriz de confusão de validação (Figura 46), que possui dados nunca antes apresentados para o modelo retreinado apresentou excelentes resultados, não errando em nenhuma das classes.

Figura 46 – Matriz de confusão do modelo MobileNet retreinado com dados de validação



Fonte – Produção do Autor

É interessante notar que, o erro pode ter ocorrido para a matriz dos dados de teste (Figura 45) por conta de dados que, mesmo sendo filtrados, apresentam características muito diferentes das levadas em consideração pelo modelo, o que acaba não ocorrendo nos dados do conjunto de validação (Figura 46).

4.1.2 Aplicação do recurso assistivo desenvolvido

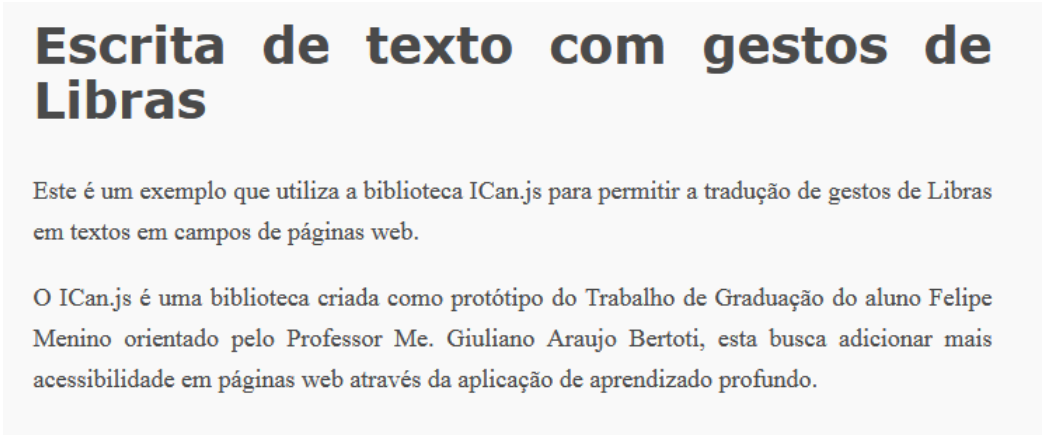
Além da validação realizada nos resultados do MobileNet utilizado neste RA, uma aplicação[1] foi desenvolvida para validar a eficácia da biblioteca na distribuição de um RA que permite a escrita de textos em campos de páginas *web*. Nesta aplicação espera-se que a biblioteca permita a escrita através de Libras em um campo de uma página *web*.

4.1.2.1 Página inicial

Esta aplicação possui uma página, em seu início (Figura 47), há uma descrição dos objetivos da página e também sobre o projeto desenvolvido.

[1] Disponível em: <https://icanjs-examples.netlify.com/escrita-de-texto/>

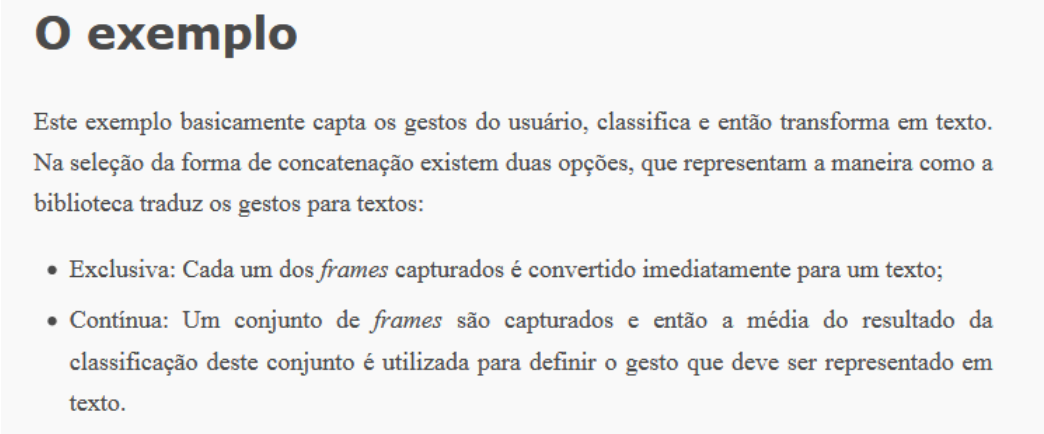
Figura 47 – Primeira parte da página da aplicação desenvolvido



Fonte – Produção do Autor

Após a parte de apresentação, há uma seção que explica as formas de funcionamento (Figura 48) do exemplo, nessa são descritas as duas formas de funcionamento descritas durante o Capítulo 3, a primeira (Exclusiva) transcrevendo o gesto capturado em cada *frame* e a segunda (Contínua), que considera os *frames* capturados em um intervalo de tempo para definir o gesto que deve ser transcrito.

Figura 48 – Seção de explicações sobre o funcionamento do exemplo da aplicação



Fonte – Produção do Autor

Com o final das explicações o exemplo presente na aplicação é apresentado, disponibilizando as duas formas de funcionamento explicadas na Figura 48. A primeira forma de funcionamento não permite a geração de um texto contínuo, transcrevendo gesto a gesto para o campo da página (Figura 49).

Figura 49 – Exemplo de funcionamento exclusivo

Fonte – Produção do Autor

Já a segunda forma de funcionamento, presente na Figura 50, permite a escrita de textos contínuos, utilizando intervalos de tempo, o que também deixa seu uso agradável.

Figura 50 – Exemplo de funcionamento contínuo

Fonte – Produção do Autor

Com isso, o objetivo da escrita de textos com Libras em campos de páginas *web* foi atingido através do módulo implementado na biblioteca.

4.2 Controle do cursor do mouse com movimentos da cabeça

Esta seção apresenta a aplicação criada com os resultados obtidos na implementação do ICan.js e suas funcionalidades para o recurso assistivo de controle de *cursor*.

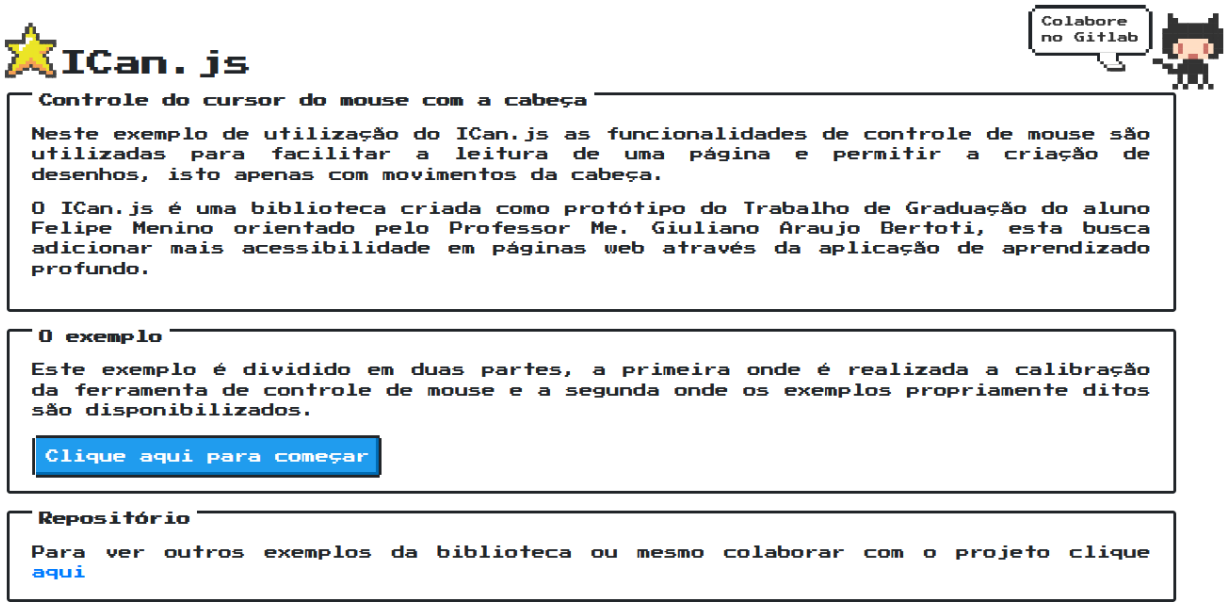
4.2.1 Aplicação do recurso assistivo desenvolvido

Para realizar testes com o recurso assistivo implementado na biblioteca, fez-se a criação de uma aplicação[2]que consome os recursos da biblioteca. Nas subseções abaixo as páginas desenvolvidas serão apresentadas.

4.2.1.1 Página inicial e calibração

A página inicial da aplicação desenvolvida possui informações gerais do projeto e também da forma de funcionamento do mesmo, como exibido na Figura 51.

Figura 51 – Tela inicial

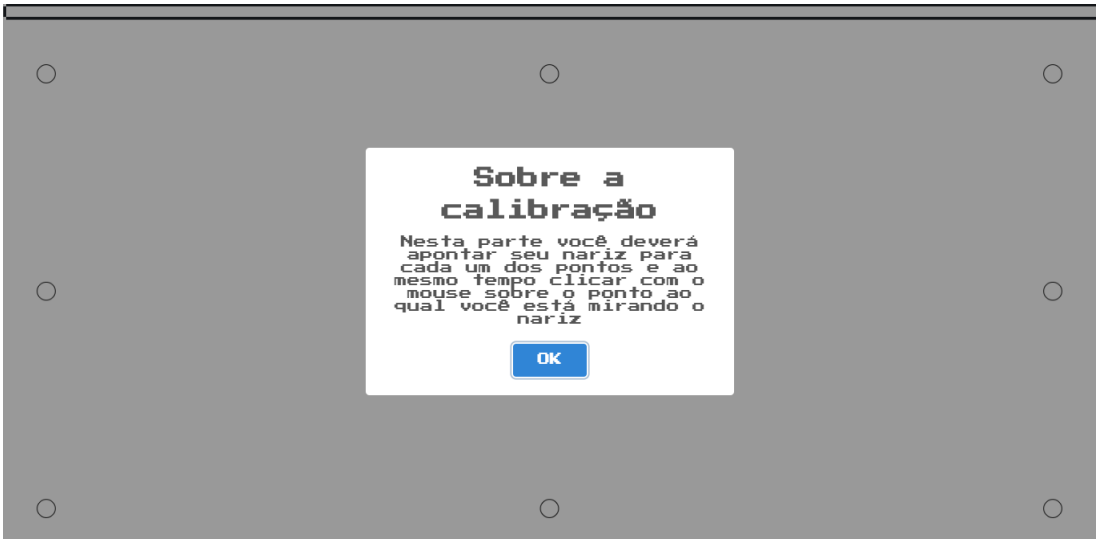


Fonte – Produção do Autor

Na Figura 51, logo após a explicação do exemplo de execução da aplicação há um botão para que a execução seja iniciada, no momento em que o usuário clica no botão, a tela de calibração do modelo de regressão que será utilizado é exibida (Figura 52), nesta primeira tela de calibração, informações gerais são exibidas e então o usuário pode começar a calibração.

[2] Disponível em: <https://icanjs-examples.netlify.com/controle-de-mouse/>

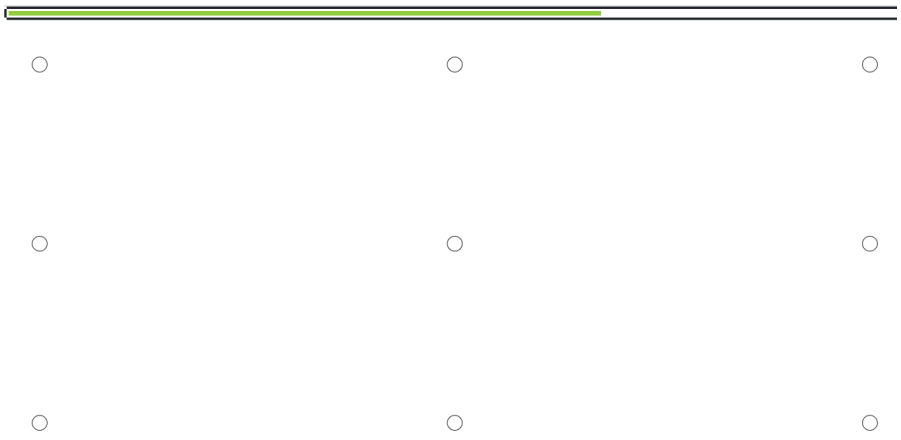
Figura 52 – Tela inicial de calibração



Fonte – Produção do Autor

Com o usuário iniciando a aplicação, fica disponível para ele uma matriz 3X3 que ao ser clicada, salva a posição do *cursor* e também do nariz do usuário. Aqui é esperado que, quando o usuário clicar nos pontos da matriz ele esteja apontando o nariz para a direção do ponto. Cada clique válido do usuário, faz uma barra de progresso no topo da tela ser atualizada (Figura 53).

Figura 53 – Matriz de calibração e barra de progresso



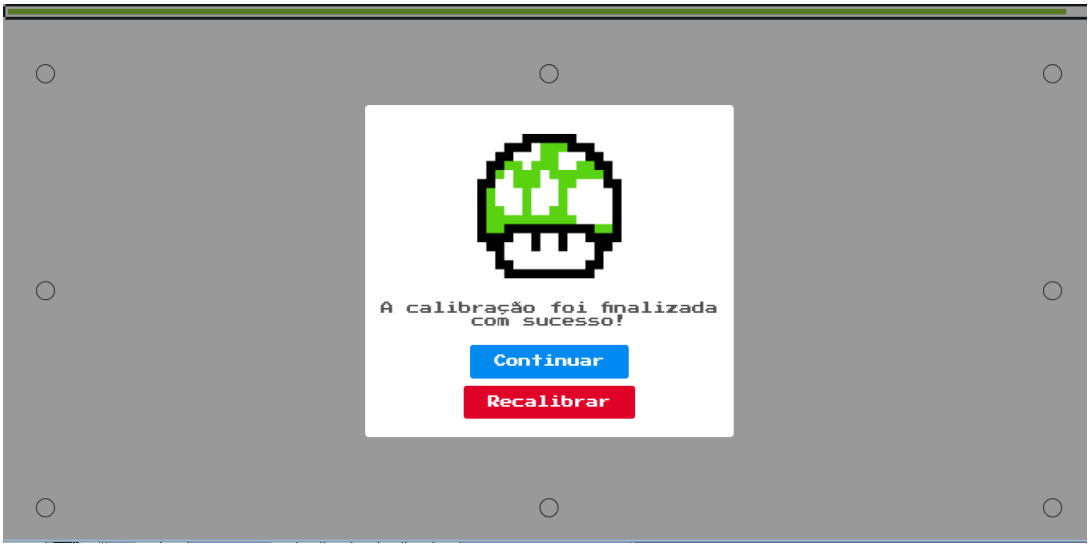
Fonte – Produção do Autor

Ao término da coleta de pontos, o modelo é calibrado e salvo no *sessionStorage*[3] do navegador, ao mesmo tempo que, uma mensagem de finalização é exibida (Figura 54),

[3] Local para armazenamento temporário em navegadores

nesta mensagem o usuário pode escolher continuar para a próxima tela ou mesmo recalibrar o modelo, sendo que, ao clicar nesta segunda opção, o processo visto anteriormente é recomeçado.

Figura 54 – Aviso da finalização da calibração

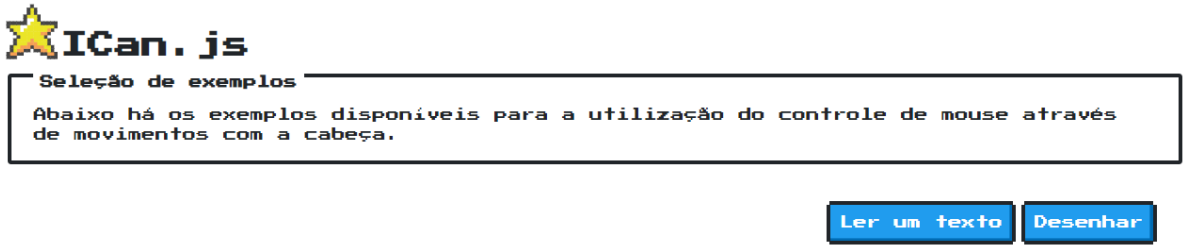


Fonte – Produção do Autor

4.2.1.2 Páginas de exemplos

Com a finalização da calibração é exibido ao usuário uma página para seleção de exemplos (Figura 55), nesta está disponível uma página para testar a funcionalidade de *scrolling* de páginas, isto feito através da leitura de um texto e outra para desenhar, que aproveita os recursos da camada *Core* para construir uma aplicação de desenhos através de gestos.

Figura 55 – Página de seleção de exemplos

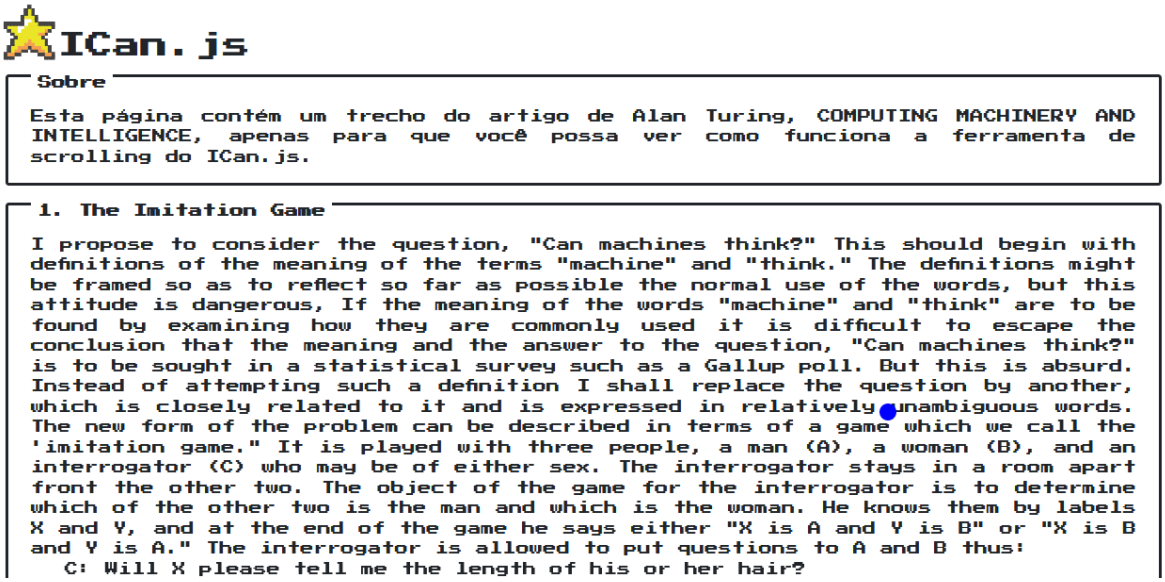


Fonte – Produção do Autor

A página de leitura de texto (Figura 56) utiliza a função *screenScroller*, sem

nenhuma mudança, o resultado é uma *div* que representa o *cursor* criado que é movimentado através dos gestos.

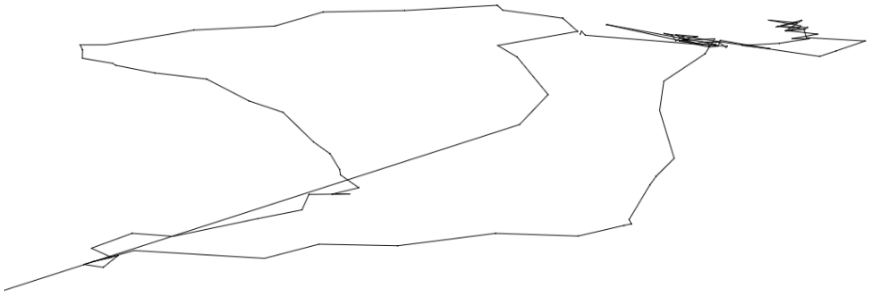
Figura 56 – Página de leitura de texto com a função *screenScroller*



Fonte – Produção do Autor

Já na página de desenho, foi criado uma nova forma de aplicação das funcionalidades presentes do ICan.js, presentes na camada *Core*, demonstrando que, além dos recursos já implementados na camada *Common* é possível realizar a implementação de novas funcionalidades com a biblioteca.

Figura 57 – Canvas de desenho criado no exemplo de desenho com o ICan.js



Fonte – Produção do Autor

Com isto, o objetivo da criação de um recurso assistivo para controle de *cursor* com gestos do usuário foi atingido através das funcionalidades implementadas no ICan.js.

5 Considerações Finais

Neste capítulo as contribuições e conclusões deste Trabalho são apresentadas, assim como as publicações e os trabalhos futuros.

5.1 Contribuições e conclusões

As contribuições apresentadas neste trabalho são:

1. Um modelo de RNC criado sob o MobileNet capaz de classificar gestos de Libras;
2. Integração do funcionamento de dois algoritmos para a produção de predições;
3. Biblioteca JavaScript que fornece recursos assistivos desenvolvidos com técnicas de Aprendizado Profundo.

Neste trabalho foi apresentado uma biblioteca para a linguagem JavaScript que fornece dois recursos assistivos, o controle de *cursor* através de movimentos com a cabeça e a escrita de texto em campos com gestos de Libras, ambos para ambiente *web*.

Para o desenvolvimento do recurso assistivo de controle de *cursor* foram utilizados o modelo de RNC PoseNet junto a regressões lineares, o que mesmo dependendo de uma calibração permitiu aos usuários a navegação dentro de uma página sem a necessidade de utilizar um *mouse*.

No desenvolvimento do recurso assistivo de escritas com sinais de Libras fez-se a utilização do MobileNet com a técnica de aprendizado por transferência no treinamento, que utilizou um conjunto de dados criados neste trabalho. A métrica utilizada para a avaliação foi a matriz de confusão, que demonstrou que o modelo apresentou bons resultados, tanto no treino e teste quanto na validação.

Todas as funcionalidades e o ecossistema criado para o desenvolvimento do ICan.js junto aos recursos assistivos citados anteriormente podem ser utilizado como base para a criação de outros recursos assistivos, e ainda, a forma de implementação da biblioteca e as técnicas utilizadas nela permitiram a criação de recursos assistivos mais acessíveis, que para serem utilizados dependem exclusivamente de tecnologias já presentes no dia-a-dia das pessoas, não sendo necessário a aquisição de nenhum *hardware* ou *software* específico.

5.1.1 Publicações

Como resultado deste trabalho foram publicados os seguintes artigos em periódicos:

1. *Deep Learning* aplicado na conversão de Libras em texto - (MENINO; BERTOTI, 2018a);

2. Uso de *Deep Learning* para desenvolver tecnologias assistivas de baixo custo - (MENINO; BERTOTI, 2018b).

5.2 Trabalhos futuros

Os resultados deste trabalho não encerram as pesquisas sobre recursos assistivos na *web* utilizando técnicas de Aprendizado Profundo, mas abrem oportunidades para os seguintes trabalhos futuros:

- Adicionar outros modelos de regressão;
- Implementar método que permita ao usuário clicar na tela no recurso assistivo de controle de *cursor*;
- Melhorar a movimentação do *cursor* com predições extras, estas relacionadas aos locais onde o usuário deseja alcançar na tela;
- Aumentar a quantidade de gestos de Libras disponíveis no ICan.js;
- Incrementar os pontos de referência para a movimentação do *cursor*.

Referências

ABADI, Martín; AGARWAL, Ashish; BARHAM, Paul; BREVDO, Eugene; CHEN, Zhifeng; CITRO, Craig; CORRADO, Greg S.; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; GOODFELLOW, Ian; HARP, Andrew; IRVING, Geoffrey; ISARD, Michael; JIA, Yangqing; JOZEFOWICZ, Rafal; KAISER, Lukasz; KUDLUR, Manjunath; LEVENBERG, Josh; MANé, Dan; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek; OLAH, Chris; SCHUSTER, Mike; SHLENS, Jonathon; STEINER, Benoit; SUTSKEVER, Ilya; TALWAR, Kunal; TUCKER, Paul; VANHOUCKE, Vincent; VASUDEVAN, Vijay; VIéGAS, Fernanda; VINYALS, Oriol; WARDEN, Pete; WATTENBERG, Martin; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>. Citado na página 16.

AMIDI, Shervine. *Convolutional Neural Networks cheatsheet*. [S.l.], 2018. Citado na página 30.

ANDRIOLI, MARY. Desenvolvimento de recursos na área de Tecnologia Assistiva: desafios e possibilidades em Institutos Federais. *Teses.Usp.Br*, n. 6, p. 278, 2017. Disponível em: <http://www.teses.usp.br/teses/disponiveis/48/48134/tde-31072017-160236/en.php>. Citado 2 vezes nas páginas 15 e 19.

ARAÚJO, Flávio H. D; CARNEIRO, Allan C; SILVA, Romuere R. V; MEDEIROS, Fátima N. S.; USHIZIMA, Daniela M. Redes Neurais Convolucionais com Tensorflow: Teoria e Prática. *III Escola Regional de Informática do Piauí*, p. 382–406, 2017. Citado 4 vezes nas páginas 28, 29, 30 e 31.

BARANAUSKAS, José Augusto. *Aprendizado de Máquina Conceitos e Definições*. 2007. Citado na página 21.

BERSCH, Rita. Introdução À Tecnologia Assistiva. 2017. Disponível em: <www.assistiva.com.br>. Citado 3 vezes nas páginas 15, 19 e 20.

BEZERRA, Eduardo. Introdução à Aprendizagem Profunda. In: _____. [S.l.: s.n.], 2016. p. 57–86. Citado na página 26.

BISHOP, Christopher. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. ISSN 0022-3263. ISBN 9780387310732. Citado na página 31.

CARVALHO, Paulo Vaz de. *Breve História dos Surdos no Mundo (Portuguese Edition)*. [s.n.], 2007. ISBN 9899525413. Disponível em: <https://www.xarg.org/ref/a/B01JLVZQUW/>. Citado na página 19.

CHOLLET, François. *keras*. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>. Citado 2 vezes nas páginas 16 e 32.

CINTRA, Rosangela. Introdução à Neurocomputação. *ELAC 2019*, p. 22, 2019. Citado 2 vezes nas páginas 21 e 24.

DICIONáRIO da Língua Brasileira de Sinais V3 - 2011. 2011. Disponível em: <http://www.acessibilidadebrasil.org.br/libras_3/>. Citado na página 38.

FRID-ADAR, Maayan; DIAMANT, Idit; KLANG, Eyal; AMITAI, Michal; GOLDBERGER, Jacob; GREENSPAN, Hayit. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing*, v. 321, p. 321–331, 2018. ISSN 18728286. Citado na página 16.

GIBSON, Adam; PATTERSON, Josh. *Deep Learning*. [S.l.]: O'Reilly Media, Inc., 2017. Citado na página 27.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 3 vezes nas páginas 26, 27 e 28.

GOOGLE. *Google Trends*. 2019. Disponível em: <https://trends.google.com/trends/explore?date=2012-01-01%202019-01-01&q=deep%20learning>. Citado na página 16.

GRIFFIN, Jonathan; RAMIREZ, Andrea. Convolutional Neural Networks for Eye Tracking Algorithm. 2018. Citado na página 49.

GUILHERMANO, Livia; CIOCARI, Antonio; AZEVEDO, Beto; TAVARES, Rogerio. Tecnologias assistivas. *TVE RS*, 2016. Disponível em: <https://www.youtube.com/watch?v=-i9Av0gfzFI>. Citado 2 vezes nas páginas 15 e 19.

HAYKIN, Simon. Redes Neurais - Princípios e prática. v. 2, p. 901, 2001. Citado 6 vezes nas páginas 21, 22, 23, 24, 26 e 30.

HODGKIN, A L; HUXLEY, A F. A QUANTITATIVE DESCRIPTION OF MEMBRANE CURRENT AND ITS APPLICATION TO CONDUCTION AND EXCITATION IN NERVE. v. 117, p. 500–544, 1952. ISSN 09237984. Citado na página 22.

HOWARD, Andrew G; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; ADAM, Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado 2 vezes nas páginas 17 e 31.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. USA: Curran Associates Inc., 2012. (NIPS'12), p. 1097–1105. Disponível em: <http://dl.acm.org/citation.cfm?id=2999134.2999257>. Citado 2 vezes nas páginas 27 e 29.

LECUN, Yann; BOTTOU, Léon; BENGION, Yoshua; HAFFNER, Patrick. Gradient-Based Learning Applied to Document Recognition. 1998. Disponível em: <http://ieeexplore.ieee.org/document/726791/{\#}full-text-sect>. Citado 2 vezes nas páginas X e 28.

MAGALHãES, Gabriel Ilharco. RECONHECIMENTO DE GESTOS DA LÍNGUA BRASILEIRA DE SINAIS ATRAVÉS DE REDES NEURAIS. 2018. Citado 4 vezes nas páginas 16, 37, 38 e 48.

MANN, Prem s. *Introdução à Estatística*. fifth. [S.l.]: LTC, 2006. Citado na página 20.

MARKEWICZ, Paula Maria. *Cumprimentos em Libras Fáceis para você Fazer – Com Vídeos*. 2017. Disponível em: <http://www.cursodelibras.org/artigos/cumprimentos-em-libras/>. Citado na página 20.

McCulloch, Warren; Pitts, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. v. 5, 1943. Citado na página 22.

MENINO, Felipe Carlos; BERTOTI, Giuliano A. Deep learning aplicado na conversÃo de libras em texto. *Bol. Técn. FATEC*, v. 46, p. 120, 2018. Citado na página 68.

MENINO, Felipe Carlos; BERTOTI, Giuliano A. Uso de deep learning para desenvolver tecnologias assistivas de baixo custo. *Bol. Técn. FATEC*, v. 46, p. 144, 2018. Citado na página 68.

MIYAZAKI, Caio Kioshi. Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D. 2017. Citado na página 28.

MURPHY, Kevin P. *Machine Learning: A Probabilistic Perspective. Adaptive Computation and Machine Learning*. [S.l.]: MIT press, 2012. Citado na página 26.

NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted boltzmann machines. p. 807–814, 2010. Citado na página 29.

NELSON, David Michael Quirino. Uso de redes neurais recorrentes para previsão de séries temporais financeiras. p. 55, 2017. Citado na página 25.

NG, Andrew; KATANFOROOSH, Kian; MOURRI, Younes Bensouda. *Neural Networks and Deep Learning*. 2016. Citado na página 21.

NIELSEN, Michael. *Neural Networks and Deep Learning*. 2018. Disponível em: <http://neuralnetworksanddeeplearning.com/chap6.html>. Citado 2 vezes nas páginas 29 e 30.

Oliveira, Fátima Inês Wolf De; CARDOSO, Luciana Santana. Recursos de tecnologia assistiva para alunas com surdez: sugestões compartilhadas por uma bolsista Pibid. *Polyphonía*, v. 22, n. 1, 2011. ISSN 2236-0514. Citado na página 19.

OSÓRIO, Fernando. Redes Neurais - Aprendizado Artificial. *I Forum De Inteligência Artificial*, n. 1, p. 1–32, 1999. Disponível em: <http://osorio.wait4.org/oldsite/IForumIA/fia99.pdf>. Citado na página 26.

OVED, Dan. *Real-time Human Pose Estimation in the Browser with TensorFlow.js*. 2018. Disponível em: <https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5>. Citado 2 vezes nas páginas 17 e 32.

PAPANDREOU, George; ZHU, Tyler; CHEN, Liang-Chieh; GIDARIS, Spyros; TOMPSON, Jonathan; MURPHY, Kevin. Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. *CoRR*, abs/1803.08225, 2018. Disponível em: <http://arxiv.org/abs/1803.08225>. Citado na página 31.

PAPANDREOU, George; ZHU, Tyler; KANAZAWA, Nori; TOSHEV, Alexander; TOMPSON, Jonathan; BREGLER, Chris; MURPHY, Kevin P. Towards accurate multi-person pose estimation in the wild. *CoRR*, abs/1701.01779, 2017. Disponível em: <http://arxiv.org/abs/1701.01779>. Citado na página 31.

PAPOUTSAKI, Alexandra; LASKEY, James. WebGazer : Scalable Webcam Eye Tracking Using User Interactions. 2016. Citado 3 vezes nas páginas 49, 51 e 52.

PASZKE, Adam; GROSS, Sam; CHINTALA, Soumith; CHANAN, Gregory; YANG, Edward; DEVITO, Zachary; LIN, Zeming; DESMAISON, Alban; ANTIGA, Luca; LERER, Adam. Automatic differentiation in pytorch. 2017. Citado na página 16.

PAVLOVSKY, Vojtech. *Introduction To Convolutional Neural Networks*. 2017. Disponível em: <https://www.vaetas.cz/posts/intro-convolutional-neural-networks/>. Citado na página 29.

PÉREZ, Fernando; GRANGER, Brian E. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, IEEE Computer Society, v. 9, n. 3, p. 21–29, maio 2007. ISSN 1521-9615. Disponível em: <http://ipython.org>. Citado na página 33.

PETERNELLI, Luiz Alexandre. *Regressão linear e correlação*. 2003. Disponível em: <http://www.dpi.ufv.br/{~}peternelli/inf162.www.16032004/materiais/CAPITULO9.p>. Citado na página 20.

PONTI, Moacir Antonelli; COSTA, Gabriel B. Paranhos da. *Como funciona o Deep Learning*. [s.n.], 2018. 63–93 p. ISBN 9788576694007. Disponível em: <http://arxiv.org/abs/1806.07908>. Citado 2 vezes nas páginas 15 e 31.

PRESS, Gil. 7 indicators of the state-of-artificial intelligence (ai), march 2019. *Forbes*, 2019. Disponível em: <https://www.forbes.com/sites/gilpress/2019/04/03/7-indicators-of-the-state-of-artificial-intelligence-ai-march-2019/amp/?__twitter_impression=true>. Citado na página 15.

RAMOS, OZ. Handsfree.js. In: *A platform for browsing the web totaly handsfree using head tracking*. [s.n.], 2019. Disponível em: <https://glitch.com/edit/#!/handsfree-starter?path=README.md:1:0>. Citado na página 16.

RAWAT, Waseem; WANG, Zenghui. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, v. 29, p. 1–98, 2017. Citado na página 30.

REMES, Chrystian. *Caracterização Por Simulação Numérica de Painéis Fotovoltaicos e Método de Rastreamento do Máximo Ponto de Potência Baseado em Redes Neurais Artificiais*. 2016. Citado na página 22.

ROSSUM, Guido Van. *Python Programming Language*. 1995. Disponível em: <https://www.python.org/>. Citado na página 32.

RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATHY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael; BERG, Alexander C.; FEI-FEI, Li. ImageNet Large Scale Visual Recognition

Challenge. *International Journal of Computer Vision*, v. 115, n. 3, p. 211–252, 2015. ISSN 15731405. Citado na página 44.

SHANKAR, Devashish; NARUMANCHI, Sujay; ANANYA, H A; KOMPALLI, Pramod; CHAUDHURY, Krishnendu. Deep Learning based Large Scale Visual Recommendation and Search for E-Commerce. 2017. ISSN 0148-0227. Disponível em: <http://arxiv.org/abs/1703.02344>. Citado na página 27.

SHEPHERD, Gordon M. The synaptic organization of the brain. Oxford University Press, 1990. Citado na página 22.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. *Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas. Fundamentos Teóricos e Aspectos Práticos*. second. [S.l.]: Artliber, 2016. Citado 4 vezes nas páginas 22, 24, 25 e 26.

SILVA, Larrisa Camila Ferreira da. Modelo de Transferência de Aprendizagem baseado em Regressão Linear Regularizada. 2017. Citado 2 vezes nas páginas 15 e 26.

SMILKOV, Daniel; THORAT, Nikhil; ASSOGBA, Yannick; YUAN, Ann; KREEGER, Nick; YU, Ping; ZHANG, Kangyi; CAI, Shanqing; NIELSEN, Eric; SOERGEL, David; BILESCHI, Stan; TERRY, Michael; NICHOLSON, Charles; GUPTA, Sandeep N.; SIRAJUDDIN, Sarah; SCULLEY, D.; MONGA, Rajat; CORRADO, Greg; VIÉGAS, Fernanda B.; WATTENBERG, Martin. Tensorflow.js: Machine learning for the web and beyond. *CoRR*, abs/1901.05350, 2019. Disponível em: <http://arxiv.org/abs/1901.05350>. Citado 4 vezes nas páginas 16, 17, 33 e 35.

TAN, Pang-Ning; STEINBACH, Michel; KUMAR, Vipin. *Introdução ao Data Mining. Mineração de Dados (Em Portuguese do Brasil)*. [S.l.]: Ciência Moderna, 2009. ISBN 8573937610. Citado 2 vezes nas páginas 20 e 21.

TONOLLI, José; BERSCH, Rita. *O que é Tecnologia Assistiva?* 2017. Disponível em: <http://www.assistiva.com.br/tassistiva.html>. Citado na página 19.

VARGAS, Ana Caroline; PAES, Aline; VASCONCELOS, Cristina Nader. Um Estudo sobre Redes Neurais Convolucionais e sua Aplicação em Detecção de Pedestres. p. 4, 2016. Disponível em: <http://sibgrapi.sid.inpe.br/col/sid.inpe.br/sibgrapi/2016/09.12.15.44/doc/um-estudo-sobre.pdf>. Citado 3 vezes nas páginas 16, 28 e 30.

WINSTON, Patrick Henry. *Artificial Intelligence*. [S.l.]: Addison-Wesley, 1992. v. 3. Citado na página 21.

ZUBEN, Fernando J. Von. Introdução à Inteligência Artificial. p. 1–48, 2013. Citado na página 21.