COORDENADORIA DE ENGENHARIA DA COMPUTAÇÃO

FELIPE DA SILVA PEREIRA

RODOLFO LUIZ CUGLER PEREIRA

RECONHECIMENTO DE LIBRAS COM CNTK E REALSENSE

Sorocaba/SP

2017

Felipe da Silva Pereira
Rodolfo Luiz Cugler Pereira

RECONHECIMENTO DE LIBRAS COM CNTK E REALSENSE

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia de Sorocaba, como exigência parcial para obtenção do Diploma de Graduação em Engenharia da Computação.
Orientador(a): Prof. Me. Fábio Lopes Caversan

Sorocaba/SP
2017

FICHA CATALOGRÁFICA

ELABORADA PELA "BIBLIOTECA FACENS"

P436r

Pereira, Felipe da Silva.

Reconhecimento de LIBRAS com CNTK e Realsense / por Felipe da Silva Pereira, Rodolfo Luiz Cugler Pereira.– Sorocaba, SP: [s.n.], 2017.

64 f.; 29cm.

Trabalho de Conclusão de Curso (Graduação) – Faculdade de Engenharia de Sorocaba, Coordenadoria de Engenharia da Computação – Curso de Engenharia da Computação, 2017.

Orientador(a): Prof.(a) Me. Fábio Lopes Caversan

1. LIBRAS. 2. CNTK. 3. RealSense. I.Pereira, Rodolfo Luiz Cugler. II.Faculdade de Engenharia de Sorocaba III. Título.

CDD 621.39

RECONHECIMENTO DE LIBRAS COM CNTK E REALSENSE

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia de Sorocaba, como exigência parcial para obtenção do Diploma de Graduação em Engenharia da Computação.

Comissão examinadora:

Me. Fábio Lopes Caversan

Victor Narcizo de Oliveira Neto

Dr. Glauco Todesco

Coordenador(a): Prof.ª Dra. Andréa Lucia Braga Vieira Rodrigues

Sorocaba/SP

2017

Dedico este trabalho primeiramente aos meus pais, pois foi graças a eles que pude chegar neste momento, e dedico também ao meu colega, Rodolfo Pereira, pelo seu comprometimento e esforço realizado durante todo o período do TCC.

Aos meus pais que estiveram sempre ao meu lado, me auxiliando e proporcionando um excelente estudo e a minha namorada, Gabriela, que compartilhou comigo esses cinco anos de graduação e me incentivou a todo instante.

AGRADECIMENTOS

Agradeço a Deus por me capacitar e orientar; a minha família pela força, pela companhia, apoio e paciência; à minha namorada Bárbara pelo apoio; aos meus amigos e companheiros de curso, em especial ao Rodolfo Pereira, parceiro e líder durante o TCC. Minha gratidão também a todos os professores e a coordenadora, presentes em minha trajetória acadêmica.

AGRADECIMENTOS

Primeiramente a Deus por permitir o cumprimento de mais uma jornada.

Aos meus pais por sempre estarem ao meu lado e pelos esforços para me garantir um ensino de qualidade.

A minha namorada, Gabriela, por me incentivar a buscar sempre mais e traçar rotas cada vez mais altas.

A meus amigos Felipe, Bianca e Alex que estiveram ajudando em todos os trabalhos.

A meu orientador, Caversan, que acreditou em nosso trabalho e nos auxiliou durante o projeto.

Ao professor Glauco Todesco que emprestou suas câmeras para que o projeto fosse desenvolvido.

A maioria das pessoas nunca vai longe o suficiente no seu primeiro vento para descobrir que elas terão uma segunda rajada. Dê a seus sonhos tudo o que você tem e você se surpreenderá com a energia que vem de você.

William James

Pensamento crítico é a habilidade mais importante que você pode ter. Não apenas porque há várias informações erradas na rede. Há informações erradas em todo o lugar.

Vint Cerf

RESUMO

Pereira, F. S.; Pereira, R. L. C., **Reconhecimento de LIBRAS com CNTK e Realsense.** Sorocaba, 2017, 64f. Trabalho de Conclusão de Curso (Graduação) – Curso de Engenharia da Computação, Faculdade de Engenharia de Sorocaba. Sorocaba, 2017.

O Brasil possui a Linguagem Brasileira de Sinais como sua segunda língua oficial, entretanto, uma alta porcentagem de brasileiros não possui conhecimento da mesma. O presente trabalho apresenta o objetivo de tentar fazer a tradução de sinais de LIBRAS para texto por meio de imagens. Tendo como focos: captura, análise de imagens e aprendizado de máquina com redes neurais. O projeto teve um bom desempenho, o que leva a acreditar que com melhorias como ganho de performance e maior base de dados para treinamento, poderia tornar o projeto em um produto.

Palavras-chave: LIBRAS. Rede Neural. Imagem.

ABSTRACT

Pereira, F. S.; Pereira, R. L. C., **Reconhecimento de LIBRAS com CNTK e Realsense.** Sorocaba, 2017, 64f. Trabalho de Conclusão de Curso (Graduação) – Curso de Engenharia da Computação, Faculdade de Engenharia de Sorocaba. Sorocaba, 2017.

Brazil has the signal brazilian language as the second oficial language, however, a high percentage of Brazilians are not aware about this. This current project aims to try to translate signal language to text through images. There are focus in: capture, image analysis and machine learning with neural network. The project had a good interpretation, which leads to believe that improvements as a better performance and bigger database for training, could become the project as a product.

Keywords: LIBRAS. Neural Network. Image.

SUMÁRIO

1 INTRODUÇÃO 16
2 AQUISIÇÃO E PROCESSAMENTO DE IMAGENS 18
2.1 CONCEITOS BÁSICOS 18
2.1.1 História das Câmeras 18
2.1.2 Captura de Imagem 19
2.1.3 Processamento e Análise 19
2.2 PROCESSAMENTO DE IMAGENS 20
2.2.1 Tipos de Imagens 20
2.2.2 Técnicas de Processamento 21
2.3 VISÃO COMPUTACIONAL 22
2.3.1 Extração de Informação 22
2.3.2 Interpretação e Análise 23
2.4 INTEL® REALSENSE™ 24
2.4.1 Kit de Desenvolvimento de Software 25
2.4.2 Requisitos 27
3 REDES NEURAIS 29
3.1 CONCEITOS BÁSICOS 29
3.1.1 Neurônio Artificial 30
3.1.2 Treinamento 31
3.2 TIPOS DE REDE 32
3.3 PERCEPTRON 34
3.4 CNTK 36
3.4.1 Descrição da Rede 36
3.4.2 Utilização 37

4 DESENVOLVIMENTO 39
4.1 ARQUITETURA 39
4.1.1 Modelo de Domínio 40
4.1.2 Diagrama de Atividades 41
4.2 COMUNICAÇÃO ENTRE SISTEMAS 42
4.3 SINCRONIZAÇÃO DAS CÂMERAS 43
4.4 TOPOLOGIA DA REDE NEURAL 43
4.5 TREINAMENTO DO ALGORITMO 45
4.5.1 Preparação dos Dados 45
4.5.2 Treinamento da Rede 46
4.6 RESULTADOS 46
5 CONCLUSÃO 49
REFERÊNCIAS 50
APÊNDICE A – CÓDIGO PARA CAPTURA DE DADOS DA CÂMERA 55
APÊNDICE B – CÓDIGO PARA CAPTURA DE IMAGENS 57
APÊNDICE C – CÓDIGO DE SINCRONIZAÇÃO DOS DADOS DAS CÂMERAS 58
APÊNDICE D – CÓDIGO PARA CRIAR A ENTRADA PARA A REDE NEURAL 60
APÊNDICE E – CÓDIGO PARA CRIAR A REDE NEURAL 62
APÊNDICE F – CÓDIGO PARA AVALIAR OS FRAMES 64

LISTA DE FIGURAS

Figura 2.1 - Intel® RealSense™ F200 25
Figura 2.2 - Módulos da câmera Realsense™ 26
Figura 2.3 - Hierarquia de chamada de interfaces da câmera 27
Figura 2.4 - Requisitos da câmera 28
Figura 3.1 - Modelo matemático proposto por McCulloch e Pitts 30
Figura 3.2 - Funcionamento do neurônio artificial 31
Figura 3.3 - Aprendizagem por reforço 32
Figura 3.4 - Rede direta 33
Figura 3.5 - Rede neura recorrente 33
Figura 3.6 - Camadas do perceptron 34
Figura 3.7 - Padrões linearmente separáveis 35
Figura 3.8 - Padrões linearmente não separáveis 35
Figura 3.9 - Criando um modelo com o CNTK 37
Figura 4.1 - Diagrama de modelo do software 41
Figura 4.2 - Diagrama de atividades do software 42
Figura 4.3 - Comunicação via socket entre sistemas 43
Figura 4.4 - Topologia da rede neural 44
Figura 4.5 - Alfabeto de LIBRAS 47

LISTA DE ABREVIATURAS E SIGLAS

CES	*Consumer Eletronics Show*
CNTK	*Microsoft Cognitive Toolkit*
DLL	*Dynamic-Link Library*
DNN	*Deep Neural Networks*
HD	*High Definition*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*
IBGE	Instituto Brasileiro de Geografia e Estatística
INES	Instituto Nacional de Educação de Surdos
LIBRAS	Linguagem Brasileira de Sinais
LSTM	*Long Short Term Memory*
RGB	*Red, Green, Blue*
SDK	Kit de Desenvolvimento de Software
SLR	Reflex Monobjetiva
SOAP	*Simple Object Access Protocol*
TCP	*Transmission Control Protocol*

1 INTRODUÇÃO

O início da educação de surdos no Brasil se dá somente em 1857, mais de 300 anos depois de seu descobrimento, com a fundação do Instituto Nacional de Educação de Surdos (INES), na cidade do Rio de Janeiro e somente em 2002, pela Lei nº 10.436 de 24 de abril, a Linguagem Brasileira de Sinais (LIBRAS) foi oficializada (MORI, 2015).

Ainda hoje, pessoas entendem o termo surdo como um termo pejorativo, porém segundo Bisol (2011), existe uma diferença entre surdos e deficientes auditivos. Surdos são pessoas que não se consideram deficientes, eles têm sua cultura própria e a valorizam. Os deficientes auditivos seriam as pessoas contrárias a cultura da comunidade surda.

Segundo o Censo realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE) em 2010, estima-se que cerca de 9,7 milhões de brasileiros possuem deficiência auditiva (DEFICIÊNCIA, 2013).

“Aproximadamente 30% dos surdos brasileiros não sabe ler português. Os restantes 70% sabem ler português mas não têm entendimento claro desta língua” (SONZA, 2017).

A proposta idealizada neste documento é de, por meio da captação de imagens, através de câmeras capazes de analisar o movimento das mãos, realizar a tradução de LIBRAS para texto.

A aquisição de imagens, foram obtidas de duas câmeras, uma frontal que fornece os dados mais relevantes para a análise, e outra auxiliar, que utilizada para desambiguação de sinais em que a câmera principal não possui uma boa visão. Para extração de informações das imagens foram utilizados componentes do kit de desenvolvimento de software (SDK) da Intel® RealSense™. Esses dados obtidos, foram salvos em um banco de dados MongoDB.

Um serviço foi desenvolvido para receber as informações, fazer a verificação dos dados recebidos e realizar a interpretação utilizando uma rede neural recorrente. Para o desenvolvimento dessa rede neural foi utilizado o *Microsoft Cognitive Toolkit* (CNTK), conjunto de ferramentas de *deep-learning.*

A rede neural foi descrita utilizando o Python, linguagem suportada pelo CNTK para a descrição da rede neural e será treinada com os dados presentes no banco de dados.

O segundo capítulo apresenta conceitos básicos a respeito de câmeras, um breve relato sobre a história delas, técnicas de análise e processamento de imagens e informações a respeito da câmera Realsense™ F200 da Intel®.

O terceiro capítulo traz informações sobre conceitos de redes neurais e sobre o CNTK.

O quarto e quinto capítulos expõem a metodologia aplicada, seus resultados e por último a conclusão do trabalho executado.

2 AQUISIÇÃO E PROCESSAMENTO DE IMAGENS

Segundo Queiroz e Gomes (2001), o processamento de imagem é uma tarefa um tanto quanto complexa. Tudo começa com a captura de uma imagem, passando por uma fase de pré-processamento e somente após esta etapa é que será possível fazer o processamento da imagem.

Existem vários fatores, como, se é necessário a utilização de câmeras de infravermelho, a configuração do tempo do obturador, valor da distância focal, entre outros, que podem ser determinísticos na hora de analisar uma imagem a fim de reconhecer padrões, extrair objetos, etc.

2.1 CONCEITOS BÁSICOS

Segundo Marques Filho e Vieira Neto (1999), seguindo do lado externo para o lado interno de uma câmera, é possível visualizar alguns componentes como:

a) Objetivas: conjuntos de lentes que direcionam os raios de luz dos pontos abertos para um ponto central ou de um ponto central para o ângulo aberto;
b) Diafragma: tem a função de controlar a quantidade de luz que atinge a câmera;
c) Obturador: controla o tempo de exposição a luz;
d) Sensor: onde pixels sensíveis à luz captam a cena.

2.1.1 História das Câmeras

Joseph Nicephore Niepce foi, em 1793, um dos primeiros estudiosos a obter sucesso com uma fotografia, porém somente no ano de 1839 que a primeira câmera fotográfica (Daguerreótipo) foi construída e comercializada. E somente em 1991 foi criada a primeira câmera reflex monobjetiva (SLR) digital (NEMES, 2014).

Em 2014 foi lançado a câmera da Intel® RealSense™ capaz de fazer reconhecimento de gestos, faces, entre outros que serão abordados mais à frente.

2.1.2 Captura de Imagem

Resumidamente, de acordo com Villegas (2009), as câmeras são recobertas por foto sensores, ou seja, sensores sensíveis a luz que são chamados de fotodiodos. Esses fotodiodos são capazes de gerar corrente elétrica diretamente proporcional a quantidade de luz que eles são atingidos. Uma matriz de fotodiodos é capaz de gerar uma imagem em tons de cinza.

Para obter a imagem colorida, é necessário a utilização de filtros coloridos vermelho, verde e azul (RGB) que podem ser interpolados e através de algumas técnicas de pré-processamento de imagem como balanço de branco, saturação, contraste, etc. é possível chegar as imagens adquiridas pelas câmeras atuais (VILLEGAS, 2009).

2.1.3 Processamento e Análise

Após a captura de uma imagem, a próxima etapa para a interpretação da cena é o processamento, que é responsável por modificar a imagem capturada a fim de facilitar a extração de informações da mesma, removendo as informações não desejadas da imagem ou destacando as informações necessárias para a fase de análise da imagem.

O processo de análise pode utilizar as informações brutas da imagem processada, o que normalmente resulta em interpretações mais custosas e menos precisas, porém com um grande nível de generalização, ou pode possuir uma etapa de extração de informação antes da análise, que é focada em extrair uma determinada informação da imagem, e por isso pode ser mais afinado, melhorando desempenho e precisão, porém não pode ser generalizado facilmente. Tendo as informações necessárias extraídas da imagem, elas são processadas por um algoritmo, o qual é responsável por detectar padrões e retornar o resultado da interpretação da análise.

2.2 PROCESSAMENTO DE IMAGENS

O campo de processamento de imagens envolve o ato de, a partir de uma imagem, aplicar algoritmos a fim de melhorar sua qualidade para atingir a finalidade desejada, podendo ser para uma aplicação que necessita extrair uma determinada informação de uma imagem, com a finalidade destacar esta informação; ou para melhorar a qualidade da imagem para a interpretação humana (MARQUES FILHO e VIEIRA NETO, 1999).

O processo de escolha da(s) técnica(s) de processamento depende de vários aspectos, mas os mais relevantes são as condições em que a imagem foi gerada, a qual pode ter iluminação inadequada, ruído, entre outros problemas que degradam a qualidade da imagem, que podem ser corrigidos via software, e as informações que se desejam extrair da imagem, por exemplo, para conversão de imagem em texto, a binarização da imagem pode eliminar todas as informações irrelevantes e manter apenas o texto a ser analisado, melhorando a precisão do software.

2.2.1 Tipos de Imagens

Saber qual o tipo de imagem a ser analisada é fundamental para a escolha ou desenvolvimento do algoritmo de processamento que será utilizado, pois diferentes tipos de imagens possuem diferentes informações. A captura de uma imagem com dispositivos sensíveis a faixa de energia no espectro eletromagnético vai representar as informações do espectro, caracterizando tipos diferentes de imagem (MARQUES FILHO e VIEIRA NETO, 1999).

Além da eficiência do algoritmo de processamento, o tipo de imagem também deve ser levado em conta na hora de fazer a análise dos resultados, usando o processo de binarização em uma imagem proveniente de uma câmera comum, a imagem resultante reflete quais partes são mais iluminadas e quais partes são mais

escuras, e aplicando este mesmo filtro em uma imagem de profundidade (obtida por um Kinect ou pela câmera RealSense™) tem como resultado uma imagem que indica as regiões que estão perto ou longe da câmera.

Cada tipo de imagem possui diversas características únicas, que dependem majoritariamente do equipamento utilizado e das condições no momento da captura da imagem. Equipamentos diferentes capturam informações diferentes da cena capturada, câmeras comuns capturam a luz refletida pelos objetos no espectro visível humano, e pode ser influenciada por diversas condições como quantia de luz, velocidade e distância dos objetos na cena, foco, entre outros, já uma câmera infravermelho para visão noturna, apesar de ser afetada por vários fatores similares da câmera comum, ela não é afetada pela quantia de luz, pois essa câmera tem como característica uma fonte própria de luz ultravioleta, e apesar de também capturar a luz refletida pelo objeto, ela captura apenas um comprimento de onda, capturando assim uma imagem em tons de cinza.

2.2.2 Técnicas de Processamento

No campo de processamento de imagens, existem várias técnicas e algoritmos prontos, que podem ser voltados para a restauração da imagem, que busca compensar defeitos na imagem ocorridos durante a fase de captura ou transmissão, ou voltadas para o realce, que tem como intenção destacar os detalhes e informações presentes na imagem que são de interesse para a análise (CONCI, AZEVEDO e LETA, 2009).

A técnica de processamento pode trabalhar de várias formas, processando a imagem pixel a pixel, como é o caso da binarização ou pode trabalhar processando regiões da imagem, que pode ser feito tanto no domínio do espaço, utilizando informação de vários pixels ao mesmo tempo, ou no domínio da frequência, utilizando algoritmos de processamento de sinais, e voltando ao domínio do espaço depois de processada na frequência (MARENGONI e STRINGHINI, 2008).

Dentre as várias técnicas existentes, a detecção de bordas será a mais explorada durante o trabalho, ela consiste em detectar na imagem mudanças bruscas de padrão, que representam bordas, como a sobreposição de um objeto com outro, ou desenhos, e possui diversas implementações e modelos matemáticos devido a sua complexidade, pois nem sempre uma borda está claramente definida, e a sensibilidade do algoritmo pode alterar significativamente a interpretação da imagem.

2.3 VISÃO COMPUTACIONAL

Visão computacional é o campo da computação que tenta fornecer a uma máquina a capacidade de visão, concedendo a habilidade de extrair informações de objetos, como forma, posição, velocidade, etc. de forma semelhante à visão humana (RIOS, 2010).

Um bom exemplo do uso da visão computacional é o reconhecimento facial, que a partir de reconhecimento de padrões, consegue identificar olhos, nariz, boca, e outras características de um rosto, e ao identificá-las juntas em suas posições corretas, uma máquina consegue “enxergar” um rosto.

2.3.1 Extração de Informação

Como a identificação de objetos normalmente é feita utilizando reconhecimento de padrões em uma imagem, fica fácil perceber sua relação com o processamento de imagem para identificar determinado objeto em uma cena, pois de acordo com Marques Filho e Vieira Neto (1999), esta etapa procura extrair características da imagem através de descritores que permitam caracterizar com precisão os objetos a serem analisados, e é a última etapa em que a entrada ainda é uma imagem, pois sua saída é um conjunto de dados correspondente à imagem.

O processo de extração pode ser feito de diversas formas, podendo ser feita pelo próprio algoritmo responsável pela análise, como é feito por uma rede neural, a

qual normalmente possui um neurônio de entrada para imagens em tons de cinza, ou três para imagens coloridas para cada pixel da imagem, fazendo o processamento em cima da imagem bruta, sem nenhuma inteligência, ou feita de forma separada, como no reconhecimento facial, em que é extraído da imagem objetos que podem representar componentes faciais, como sobrancelha, olho, boca, entre outros, que serão interpretados na fase de análise.

2.3.2 Interpretação e Análise

O ato de analisar e interpretar uma imagem ou cena é um processo extremamente intuitivo e natural para o ser humano, ao ver uma casa, por exemplo, uma pessoa é capaz de identificá-la sem grandes esforços, pois é natural, mas para uma máquina este processo se torna complexo e trabalhoso, pois para ela inicialmente a casa não passa de um conjunto de pontos com cores. Marques Filho e Vieira Neto (1999), afirmam que o reconhecimento é o processo de classificação de um objeto baseado em suas características, e a tarefa de interpretação consiste em dar um significado a este conjunto de características em uma mesma cena.

Para uma máquina realizar a interpretação ela necessita encontrar padrões na imagem e analisá-los, isto pode ser feito de diversas formas, mas para o contexto do trabalho serão utilizadas as seguintes técnicas:

Ao extrair componentes da imagem que possuem padrões conhecidos, como formas geométricas, ou formas de objetos previamente definidos, seguido da análise destes componentes em conjunto, desta forma se uma imagem tem como objetos extraídos um quadrado grande com um retângulo dentro encostado em sua base, e um triângulo laranja logo acima com base de mesmo tamanho terá como resultado da interpretação “casa”, considerando que este padrão é conhecido.

Utilizar as informações brutas de uma imagem processada para realçar as características que se deseja analisar, ou sem nenhum processamento para uma análise mais genérica, porém custosa e menos precisa, como é o caso da análise feita

através de uma rede neural, que utiliza como entrada todos os pixels da imagem para realizar a análise, e como saída retorna o resultado da interpretação.

2.4 INTEL® REALSENSE™

"A câmera 3D Intel® RealSense™ é o primeiro módulo integrado de profundidade 3D e câmera 2D do mundo que ajuda os dispositivos a "ver" a profundidade como o olho humano" (EDEN, 2014).

A câmera veio a público pela primeira vez na *Consumer Eletronics Show* (CES) em Las Vegas em 2014 e é resultado de uma parceria entre algumas empresas como 3D Systems, Autodesk, DreamWorks, Metaio, Microsoft Skype and Lync, Scholastic, Tencent entre outras (Intel Newsroom, 2014).

A tecnologia Intel® RealSense™ é uma tecnologia que está preparada para fazer reconhecimento de imagem através de duas câmeras infravermelhas para captar a profundidade e a câmera de alta resolução capta *high definition* (HD) de 1080p para proporcionar vídeos e fotos nítidas e claras (Intel, acessado em 23 abr. 2017).

Abaixo, na figura 2.1, é possível ver uma câmera Intel® RealSense™ modelo F200 que será utilizada ao longo do projeto.

Figura 2.1 - Intel® RealSense™ F200

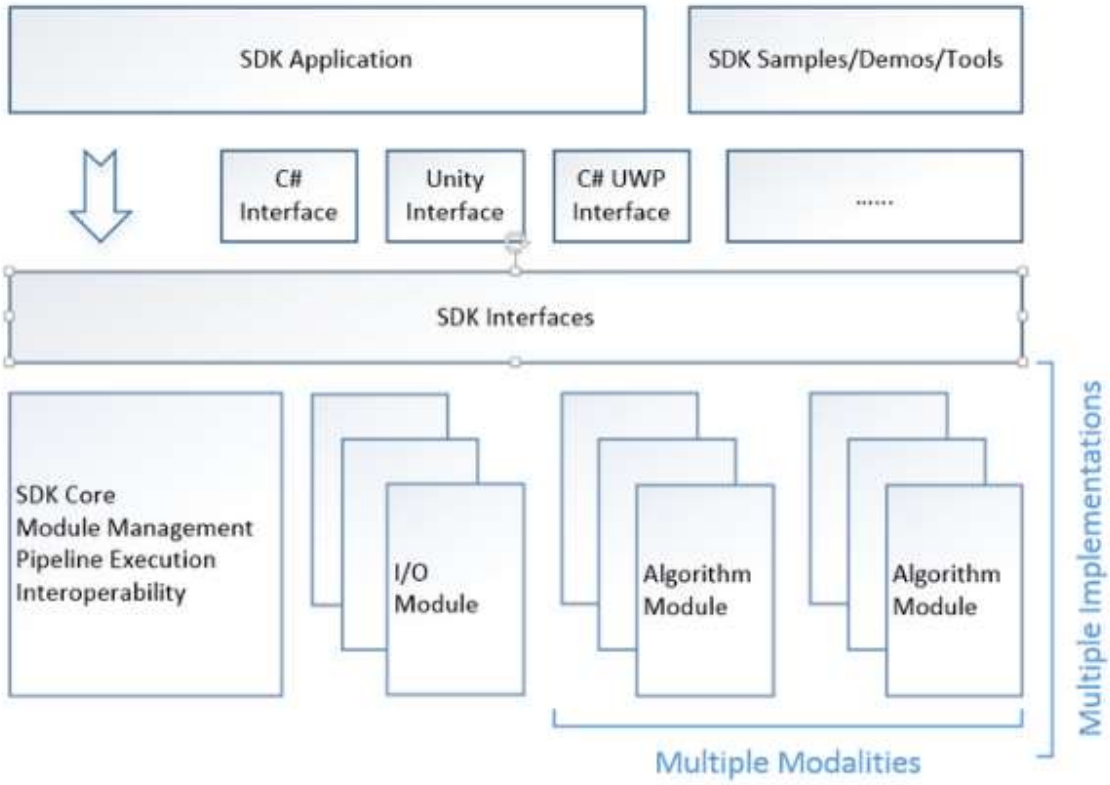
Fonte: Code Project[1]

2.4.1 Kit de Desenvolvimento de Software

A tecnologia disponibiliza um SDK que pode ser representado por módulos conforme figura 2.2.

[1] Disponível em: https://www.codeproject.com/articles/999269/using-intel-realsense-technology-in-combination-wi?msg=5147082. Acesso em: 22 abr. 2017.

Figura 2.2 - Módulos da câmera Realsense™



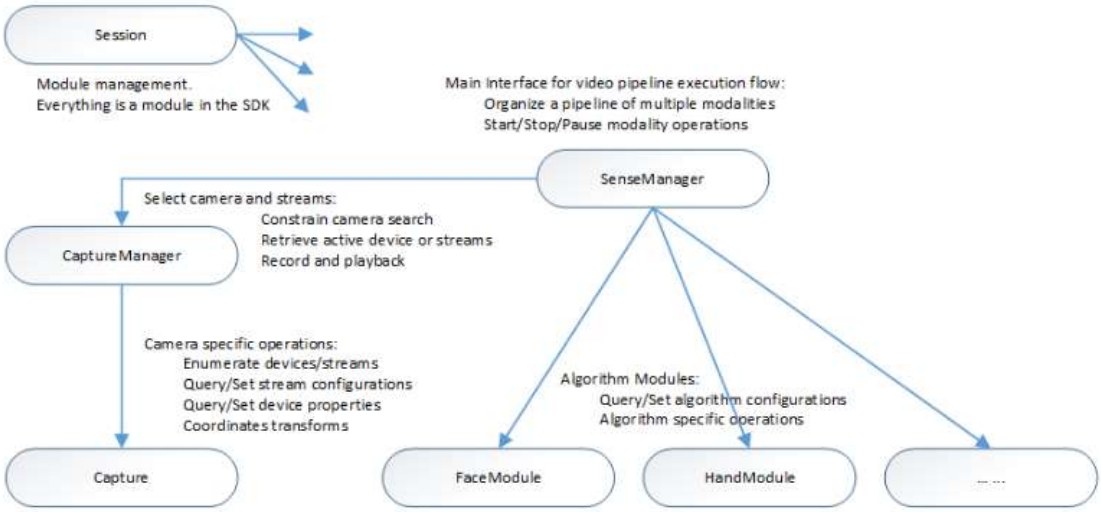
Fonte: Documentação da câmera Intel® RealSense™[2]

O módulo de *input/output* (I/O) e os módulos de algoritmos são os principais, eles são responsáveis pela captura da imagem e pelo processamento dela, sendo assim, informações tais como: reconhecimento facial, reconhecimento de gestos, reconhecimento de fala se tornam muito mais fáceis de serem adquiridos.

Abaixo segue a imagem (figura 2.3) de hierarquia de chamada de interfaces da câmera.

[2] Disponível em: https://software.intel.com/sites/landingpage/realsense/camera-sdk/v2016r3/documentation/html/index.html?doc_devguide_hardware_and_software_requirements.html. Acesso em: 23 abr. 2017.

Figura 2.3 - Hierarquia de chamada de interfaces da câmera



Fonte: Documentação da câmera Intel® RealSense™[3]

É através dessas interfaces que é possível acessar as funcionalidades da câmera e conseguir os dados que ela é capaz de retornar.

2.4.2 Requisitos

Na própria documentação da Intel é possível ver alguns requisitos básicos para o funcionamento da câmera (conforme figura número 2.4), eles podem ser úteis na definição de uma arquitetura de um sistema que utiliza a câmera.

[3] Disponível em: https://software.intel.com/sites/landingpage/realsense/camera-sdk/v2016r3/documentation/html/index.html?doc_devguide_hardware_and_software_requirements.html. Acesso em: 23 abr. 2017.

Figura 2.4 - Requisitos da câmera

Hardware requirements:

- 4th generation Intel® Core™ processors based on the Intel microarchitecture code name Haswell
- 8GB free hard disk space
- Intel® RealSense™ Camera SR300 (required to connect to a USB* 3 port)

Software requirements:

- Microsoft* Windows* 10 OS 64-bit
- Microsoft Visual Studio* 2012-2015 with the latest service pack
- Microsoft .NET* 4.0 Framework for the C# development
- Unity* PRO 5.2.3p3 or later for the Unity game development
- The following system configuration for the Universal application development:
  - Microsoft Windows 10
  - Microsoft Visual Studio 2015

Fonte: Documentação da câmera Intel® RealSense™[4]

[4] Disponível em: https://software.intel.com/sites/landingpage/realsense/camera-sdk/v2016r3/documentation/html/index.html?doc_devguide_hardware_and_software_requirements.html. Acesso em: 23 abr. 2017.

3 REDES NEURAIS

"Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência" (CARVALHO, 2017).

De acordo com Cardoso (2000), através de impulsos nervosos que passam de uma célula para outra, forma-se uma cadeia de informação dentro de uma rede de neurônios. A partir disto, foram criados modelos de neurônios artificiais simulando o seu comportamento, capazes de receber vários sinais e propagar sua resposta para outros neurônios, e a partir da junção de vários neurônios, as redes neurais.

3.1 CONCEITOS BÁSICOS

Basicamente, uma rede neural é composta por vários neurônios interligados entre si, e separados entre diversas camadas, de forma que cada camada realiza um certo processamento nos dados recebidos, e fornece uma resposta para a próxima camada.

A maneira com que são dispostas as camadas, e os neurônios dentro dessas camadas, forma a topologia da rede. De acordo com Iyoda (2000), a primeira camada possui neurônios especiais, responsáveis exclusivamente por receber dados e inseri-los na rede, sendo a única camada que não modifica os dados. A última camada possui neurônios de saída, responsáveis por fornecer a resposta da rede neural dada uma certa entrada. As demais camadas são chamadas de camadas escondidas, que fornecem uma resposta para a próxima camada de acordo com os dados recebidos da camada anterior.

Definida a topologia de uma rede neural, ela necessita ser treinada e avaliada. O treinamento da rede é feito utilizando dados conhecidos e corrigindo a rede

conforme os resultados errados, ou seja, “[...] treinamento a partir dos casos reais conhecidos, adquirindo, a partir daí a sistemática necessária para executar adequadamente o processo desejado dos dados fornecidos.” (TATIBANA e KAETSU, 2017). Já a avaliação consiste em medir a acurácia da rede de acordo com a resposta a uma entrada conhecida não utilizada no treinamento.

3.1.1 Neurônio Artificial

Zambiasi (2017) diz que McCulloch e Pitts criaram um modelo matemático para representar o funcionamento de um neurônio, considerando que o neurônio funciona de forma binária, ou seja, de acordo com as suas entradas, ele seria ativado ou não. O modelo matemático proposto foi o seguinte, conforme a figura 3.1:

Figura 3.1 - Modelo matemático proposto por McCulloch e Pitts

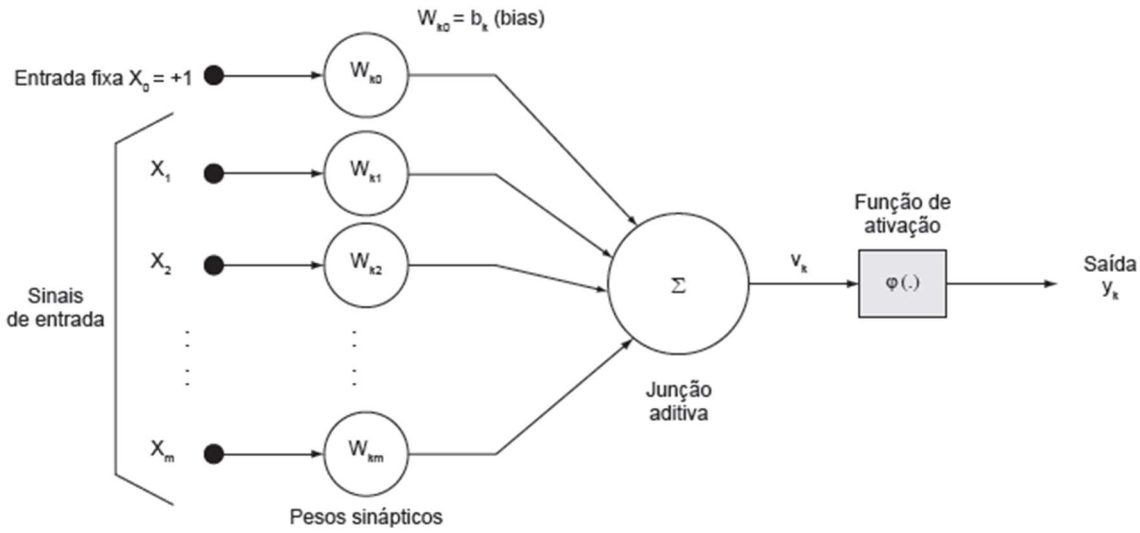
$$\varphi\left(\sum_{j=1}^{n} w_j x_j\right)$$

Fonte: Elaborado pelo autor

Onde x_j representa a j-ésima entrada de um neurônio, w_j representa um peso que multiplica essa entrada, e φ representa a função de ativação responsável por definir se a saída do neurônio será ativada ou não.

O modelo representado pela figura 3.2 é uma modificação do modelo de McChulloch e Pitts, que adiciona uma entrada denominada bias, que de acordo com Tabitana e Kaetsu (2017), “serve para aumentar os graus de liberdade, permitindo uma melhor adaptação, por parte da rede neural, ao conhecimento à ela fornecido”.

Figura 3.2 - Funcionamento do neurônio artificial



Fonte: Figura retirada do site researchgate[5]

Para que um neurônio artificial possa aprender e fornecer uma resposta de acordo com sua experiência, de forma similar ao biológico, ele necessita passar pela fase de aprendizado. De acordo com Zambiasi (2017), o aprendizado é feito a partir da seguinte equação: $w_j = w_j + \Delta w_j$, na qual o peso do neurônio é somado a um delta, calculado da seguinte maneira: $\Delta w_j = \eta e x_j$, onde η é a taxa de aprendizagem, e é o erro calculado a partir da diferença da saída desejada e da saída obtida, e x_j é a entrada do neurônio.

3.1.2 Treinamento

Segundo Zuben e Castro (2017), o treinamento de uma rede neural trata-se do ajuste dos parâmetros da rede por meio de estímulos ambientais conhecidos como padrões, afim de melhorar seu desempenho.

Existem três paradigmas de aprendizados:

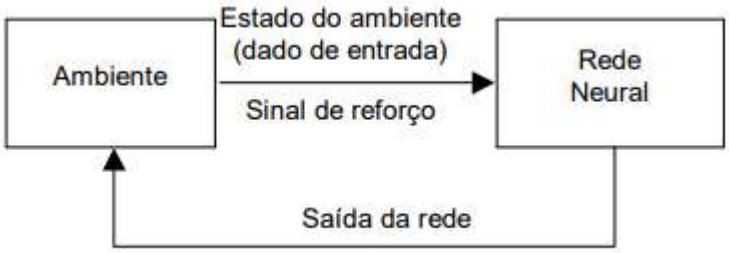
[5] Disponível em: https://www.researchgate.net/figure/281869825_fig1_Figura-14-Funcionamento-do-neuronio-artificial-baseado-em-HAYKIN-1999-p-38. Acesso em: 23 ago. 2017.

Aprendizado supervisionado: existe um professor (agente externo) que irá demonstrar os comportamentos bons e ruins (ZAMBIASI, 2017).

Aprendizado não-supervisionado: não existe agente externo para avaliar os dados de entrada (Tabitana e Kaetsu, 2017).

Aprendizagem por reforço: a cada entrada, a rede terá uma resposta de saída conforme figura 3.3, que gera um reforço para ser utilizado na próxima entrada.

Figura 3.3 - Aprendizagem por reforço



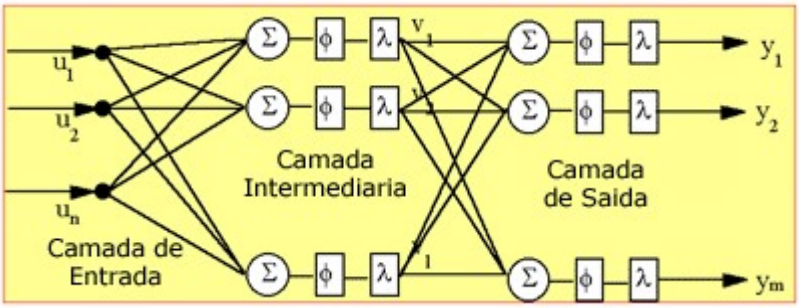
Fonte: Zuben e Cardoso, 2017

3.2 TIPOS DE REDE

“A topologia das Redes Neurais Artificiais depende da forma como os Neurônios se conectam para formar uma "Rede" de neurônios. A topologia pode ser de redes diretas (*Feedforward*) ou de redes recorrentes (*Feedback*)” (ZAMBIASI, 2017).

Segundo Zambiasi (2017), as redes diretas (figura 3.4) não possuem ciclos e são representadas na forma de camadas, onde há a camada de entrada que recebe o sinal de excitação e a camada de saída que recebe o resultado da rede.

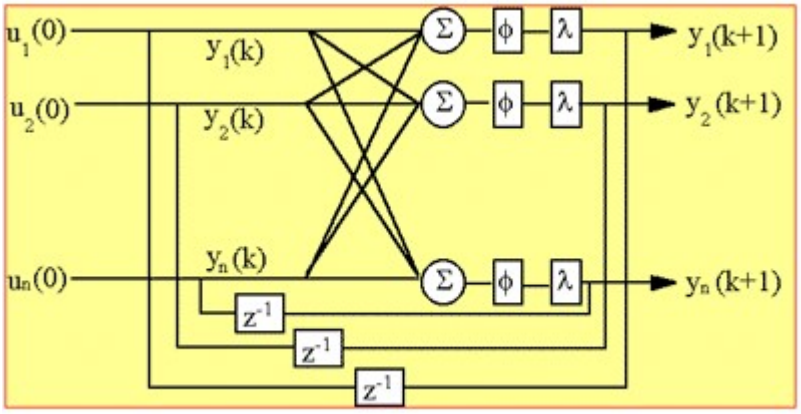
Figura 3.4 - Rede direta



Fonte: Zambiasi, 2017

As redes neuras recorrentes, contém ao menos um ciclo em suas conexões, conforme figura 3.5.

Figura 3.5 - Rede neura recorrente



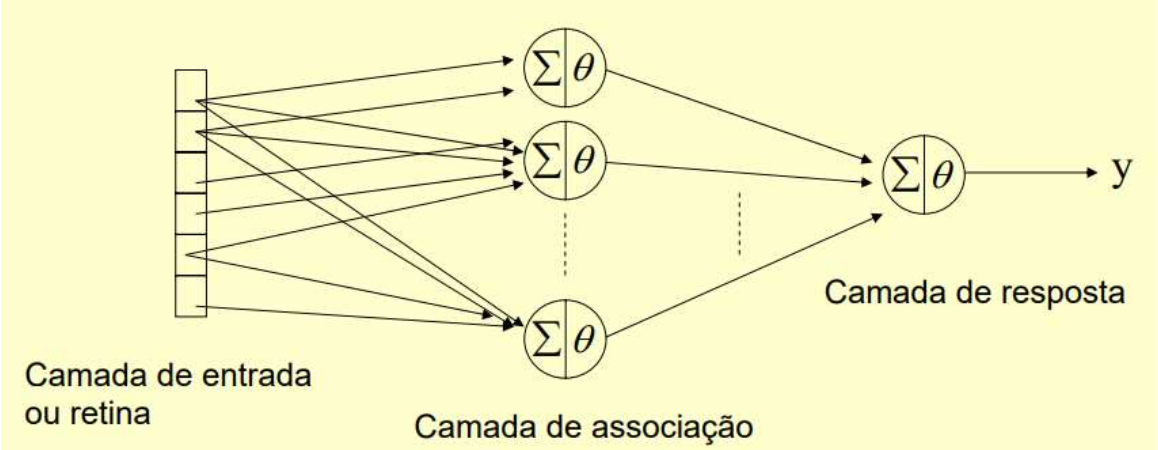
Fonte: Zambiasi, 2017

3.3 PERCEPTRON

“No final da década de 1950, Rosenblatt na Universidade de Cornell, criou uma genuína rede de múltiplos neurônios do tipo discriminadores lineares e chamou esta rede de perceptron” (GONÇALVEZ, 2004).

Segundo Thomé e Marques (2003), o perceptron é uma rede neural que possui neurônios dispostos entre a camada de entrada, camada de associação e camada de resposta, conforme demonstrado na figura 3.6.

Figura 3.6 - Camadas do perceptron

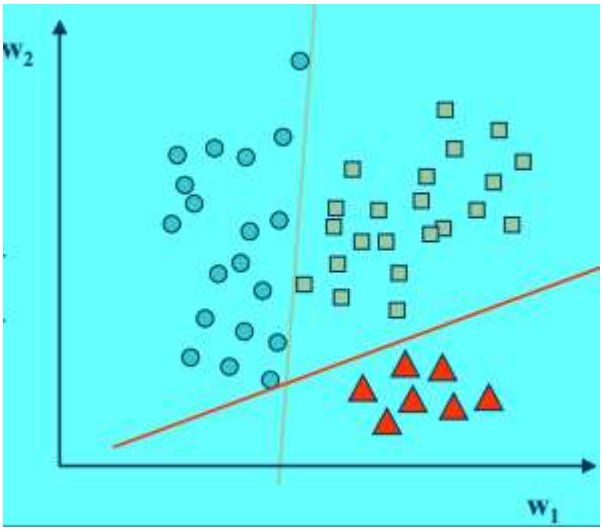


Fonte: Thomé e Marques, 2003

Para o treinamento de um perceptron, são fornecidos vetores para a rede, se a rede acerta o resultado, nada irá acontecer, porém se depois te ter passado pela função de ativação e a função de propagação e o resultado obtido for diferente do resultado esperado, o perceptron irá ajustar os pesos dos neurônios afim de chegar nos pesos ideias.

O percetron simples possui respostas somente iguais a 0 ou 1 portanto é possível representar através dele problemas linearmente separáveis conforme representado na figura 3.7 (THOMÉ, 2005).

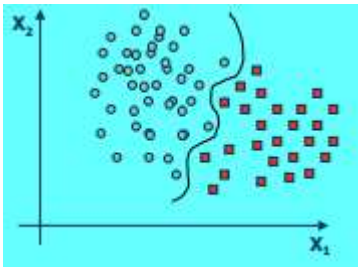
Figura 3.7 - Padrões linearmente separáveis



Fonte: Thomé, 2005

Para casos onde não é possível separar os padrões linearmente é necessário utilizar-se de modelos mais complexos do que o perceptron simples, como o perceptron multicamada ou algoritmos de *deep learning*, conforme figura 3.8.

Figura 3.8 - Padrões linearmente não separáveis



Fonte: Thomé, 2005

3.4 CNTK

"CNTK, o *Microsoft Congnitive Toolkit*, é um sistema para descrever, treinar e executar redes computacionais. Ele é também um framework para descrever máquinas de aprendizado arbitrárias como *Deep Neural Networks* (DNN)" (PYTHON, 2017).

O toolkit pode ser obtido gratuitamente no site da Microsoft, e possui versões para sistemas operacionais Windows ou Linux, com ou sem placa gráfica dedicada. Para realizar a instalação, as instruções detalhadas se encontram no site da Microsoft: https://docs.microsoft.com/en-us/cognitive-toolkit/Setup-CNTK-on-your-machine.

3.4.1 Descrição da Rede

Descrever a topologia de uma rede neural utilizando o CNTK é uma tarefa simples, caso a mesma já esteja definida. De acordo com o Microsoft (2017), pode-se definir o formato dos dados de entrada e de saída da rede neural utilizando a função *input_variable* através do parâmetro *shape*, que recebe um *int* caso seja uma entrada unidimensional, ou um *tuple* com o tamanho das dimensões.

Para descrever as camadas da rede neural, podem ser utilizadas diversas funções como *Dense*, *Convolution*, Long Short Term Memory (LSTM*)*, entre outros, onde cada uma delas é responsável por criar uma função que constrói o objeto da rede neural em si. A imagem a seguir exemplifica duas possíveis maneiras de se criar um modelo.

Figura 3.9 - Criando um modelo com o CNTK

```
def create_model_sequencial(input_var):
    with C.layers.default_options(initial_state=0.1):
        model = C.layers.Sequential([
            C.layers.Embedding(EMB_DIM, name='embed'),
            C.layers.Recurrence(C.layers.LSTM(HIDDEN_DIM), go_backwards=False),
            C.layers.Dense(NUM_LABELS, name='classify')
        ])(input_var)
        return model

def create_model(input_var):
    with C.layers.default_options(initial_state=0.1):
        model = C.layers.Embedding(EMB_DIM, name="embed")(input_var)
        model = C.layers.Recurrence(C.layers.LSTM(HIDDEN_DIM), go_backwards=False)(model)
        model = C.layers.Dense(NUM_LABELS, name='classify')(model)
    return model
```

Fonte: adaptado de Python, 2017

De acordo com Python (2017), para realizar o treinamento de uma rede, pode-se destacar dois objetos chave no processo, o *Learner* e o *Trainer*. O *Learner* é o objeto responsável por definir os parâmetros de aprendizado da rede, e pode ser instanciado de acordo com os vários construtores de acordo com o algoritmo desejado. O *Trainer* é o objeto que realiza o treino da rede, “[...] ele encapsula o processo de treinamento e emprega um ou mais *Learners* para ajustar os parâmetros de um modelo especificado [...]” (PHYTON, 2017). Com o objeto *Trainer* instanciado, podem ser iniciadas as épocas de treinamento, nas quais são fornecidos dados e o resultado esperado para que a rede seja de fato treinada. A rede treinada pode ser salva utilizando o comando *save(path)*.

3.4.2 Utilização

De acordo com CNTK (2017), uma rede neural criada pelo CNTK pode ser consumida utilizando a biblioteca CNTK *Eval*, tendo suporte para as linguagens de programação C++, Python, C# / .NET e Java, oferecendo suporte para utilização em

CPU ou GPU, múltiplas requisições paralelas de avaliação e otimização de memória ao compartilhar um modelo entre *threads*.

Para avaliação no ambiente Windows e linguagem C#, é necessário a utilização do Visual Studio Update 3, e do pacote *CNTK.GPU* para computadores com GPU compatível, ou *CNTK.CPUOnly* para os demais computadores, ambos podem ser obtidos a partir do *NuGet*.

Para carregar a rede na aplicação, utiliza-se o método *Function.Load()* passando como parâmetros o local do arquivo da rede neural e um objeto *DeviceDescriptor* com a unidade de processamento a ser utilizada. Os parâmetros de entradas e saídas podem ser obtidos do modelo carregado, e populados com os dados a serem avaliados. A avaliação ocorre ao chamar o método *Evaluate()*, que recebe os parâmetros de entrada e saída, e seu resultado obtido com o método *GetDenseData()* ou *GetOneHotData()*, de acordo com o formato desejado.

4 DESENVOLVIMENTO

A proposta idealizada neste projeto é de utilizar duas câmeras desenvolvidas pela Intel® (Realsense™ F200), posicionadas em ângulos diferentes, formando um sistema controlado com a finalidade de capturar, processar e salvar os dados de sinais feitos em LIBRAS de forma supervisionada. A partir do momento em que haja dados suficientes, será utilizado o CNTK para analisar os dados e assim realizar a tradução dos sinais realizados para texto.

Foi dado o nome de *LIBRAS Connect* ao software resultante deste projeto e nele é possível determinar três fases bem distintas, sendo elas, a captura e armazenagem dos dados, o treinamento de uma rede neural a partir dos dados salvos e o uso da rede para determinar as saídas em tempo de execução.

4.1 ARQUITETURA

A câmera utilizada no projeto foi a câmera desenvolvida pela Intel® e esta possui uma série de bibliotecas auxiliares para facilitar o trabalho do desenvolvedor, como, por exemplo, não é necessário o desenvolvedor criar funções em C/C++ para acessar a câmera e capturar os dados da imagem, para isso é necessário que o projeto importe uma biblioteca e consuma o método já desenvolvido pelo fabricante.

A câmera Realsense™ foi desenvolvida para rodar em um sistema operacional Windows e a maior parte das funções implementadas pelo fabricante nas bibliotecas são em C#, gerando as *dynamic-link libraries (DLL)*.

Por esse motivo foi escolhido como linguagem de desenvolvimento padrão o C# e o Visual Studio 2015 como ferramenta de desenvolvimento.

A quantidade de quadros geradas por segundo pelas câmeras é muito alta, em média de 30 quadros/segundo ou um quadro a cada 33 milissegundos. Esses dados necessitam ser salvos em um banco de dados e por isso a quantidade de acesso nesse banco seria tão alta quanto a geração de dados. Sendo assim, o banco escolhido para armazenamento dos dados foi o MongoDB, pela alta velocidade e a facilidade de desenvolvimento com o C#.

Para os códigos relacionados ao CNTK, como, por exemplo, a criação da rede neural, foi adotado o Python como linguagem de desenvolvimento, devido a melhor documentação e facilidade de programação.

Por último, a câmera da Intel® gera a dependência de estar conectada em um computador de forma singular, de tal maneira que as duas câmeras não podem estar conectadas no mesmo computador, assim, é necessário o uso de no mínimo duas máquinas para executar o projeto. Por se tratar de um trabalho acadêmico, ele foi projetado para funcionar em rede local, com o banco de dados e a inteligência sendo executado em uma máquina e a cada câmera sendo processada em cada computador.

4.1.1 Modelo de Domínio

O modelo de domínio, ilustrado na figura 4.1, pode ser descrito como quatro pacotes essenciais:

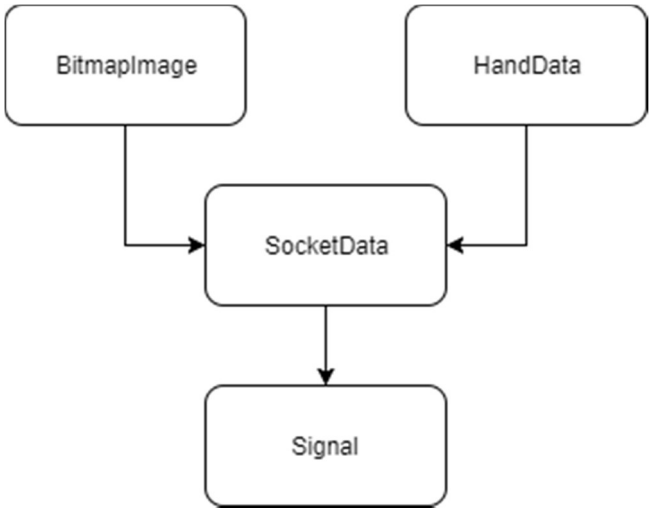
HandData: contém todos os objetos, coordenadas e outras informações, como, se é a mão direita ou esquerda, qual sua velocidade, etc. disponibilizadas pelo SDK da Intel®. A rotina de captura desses dados pode ser vista no apêndice A.

BitmapImage: proveniente da biblioteca .NET serve para armazenar as informações da imagem, como, altura, largura e todos os seus pixels. A rotina de captura dessas imagens pode ser vista no apêndice B.

SocketData: recebe os dados de *HandData* e *BitmapImage* para trafegá-los de um lado ao outro entre os computadores.

Signal: une os dados de *SocketData* no servidor fornecidos pelos dois computadores e atribui uma *label* que é fornecida pelo usuário para poder armazenar os dados em banco, montando uma base para o treinamento de uma rede neural supervisionada.

Figura 4.1 - Diagrama de modelo do software



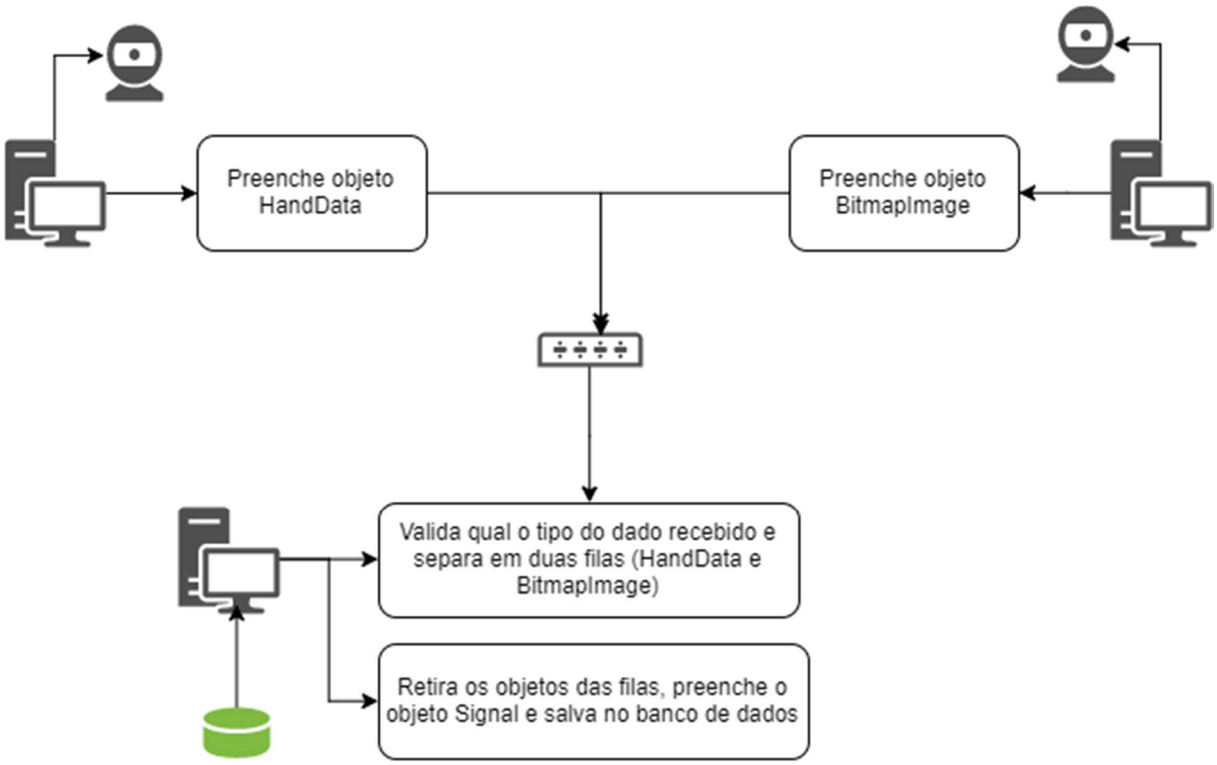
Fonte: Elaborado pelo autor

4.1.2 Diagrama de Atividades

Cada computador, denominado cliente, é responsável por um tipo de dado, sendo um computador responsável pelo *HandData* e outro computador responsável pelo *BitmapImage*. Esses objetos são serializados e enviado pela rede para um terceiro computador que foi denominado de servidor (uma das duas máquinas pode fazer o papel de cliente e servidor). O servidor irá validar qual o tipo de dado que chegou e separá-los em duas filas. Em outra *thread*, as filas serão unidas e junto com a *label* informada pelo usuário elas serão salvas no banco de dados MongoDB, finalizando assim o ciclo de armazenagem de dados.

O ciclo descrito acima foi representado na figura 4.2 abaixo.

Figura 4.2 - Diagrama de atividades do software



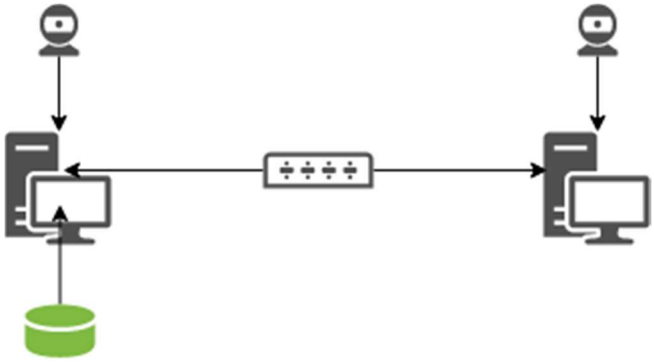
Fonte: Elaborado pelo autor

4.2 COMUNICAÇÃO ENTRE SISTEMAS

Como é necessário o uso de pelo menos duas máquinas para utilizar a aplicação é necessário um sistema de comunicação entre essas aplicações. O sistema de comunicação escolhido foi o socket, pois a segurança não é um problema no projeto, por se tratar de um ambiente fechado e o socket trabalha em cima do protocolo *Transmission Control Protocol (*TCP) e dispensa cabeçalhos extras de um protocolo *Hypertext Transfer Protocol* (HTTP) ou o envelopamento de uma aplicação *Simple Object Access Protocol* (SOAP), permitindo assim uma maior velocidade de transmissão.

A figura 4.3 abaixo representa a comunicação entre os sistemas.

Figura 4.3 - Comunicação via socket entre sistemas



Fonte: Elaborado pelo autor

4.3 SINCRONIZAÇÃO DAS CÂMERAS

O gerenciamento das filas de *HandData* e *BitmapImage* são responsáveis por fazer a sincronização das câmeras, de tal forma que as imagens só irão para o banco de dados quando houver informação proveniente das duas câmeras, ou seja, existe uma *thread* que fica validando se as duas filas possuem informação, caso sim, elas montam o objeto *Signal* e envia para o MongoDB. A rotina de sincronização dos dados de cada câmera pode ser vista no apêndice C.

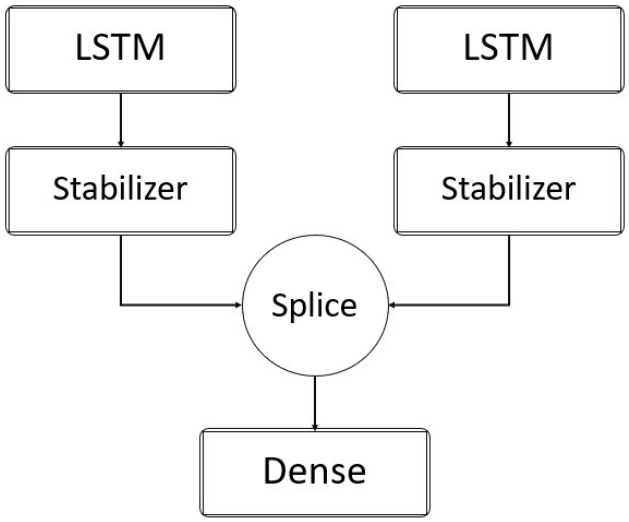
4.4 TOPOLOGIA DA REDE NEURAL

A topologia da rede neural utilizada para o reconhecimento de LIBRAS foi definida utilizando a biblioteca do CNTK para a linguagem Python. Como o reconhecimento de LIBRAS é continuo e uma letra ou palavra depende do estado anterior do gesto para ser interpretado corretamente, a primeira camada da rede

neural consiste em duas redes recorrentes do tipo LSTM, ambas com 300 neurônios, sendo que uma delas será alimentada no sentido inverso, ou seja, os primeiros itens interpretados são os últimos dados do gesto, que são seguidas por uma camada *Stabilizer*, que ajuda na velocidade do treinamento.

Em seguida, através do método *splice*, que realiza a concatenação de dois tensores, os resultados de ambas LSTM são reagrupadas e utilizadas como entrada para a camada final da rede, do tipo *Dense*, composta por N neurônios, onde N é a quantia de gestos a serem classificados, que além de ser responsável por fornecer a resposta da rede neural, também realiza sua normalização através do *Softmax* (também conhecida como função de normalização exponencial), limitando assim a saída em valores entre 0 e 1. A figura a seguir representa a topologia da rede neural utilizada.

Figura 4.4 - Topologia da rede neural



Fonte: Elaborado pelo autor

O código para criar a rede neural pode ser visto no apêndice E.

4.5 TREINAMENTO DO ALGORITMO

O treinamento da rede neural pode ser dividido em duas etapas principais:

Preparação dos dados: Leitura dos dados armazenados no banco de dados, seleção dos dados essenciais para o treinamento, e gravação no formato de leitura do CNTK.

Treinamento da Rede: Criação de instâncias dos objetos necessários para realizar o treinamento da rede, e a execução das épocas de treinamento.

4.5.1 Preparação dos Dados

A leitura dos dados armazenados no banco de dados é feita utilizando o componente *MongoClient* da biblioteca *pymongo*, que permite estabelecer uma conexão com o banco. Com a conexão aberta, é realizada uma consulta para retornar todos os dados gravados no banco de dados, os quais são salvos em uma lista.

Os dados obtidos da rede são separados em *frames*, e cada um deles contém informações da câmera comum, da RealSense™ e a palavra que aquele frame representa. Para cada *frame* é construído um objeto que contém a palavra que está sendo representada e uma lista composta pela concatenação das informações extraídas da RealSense™ com as da câmera comum, o qual é então adicionado em uma lista que será retornada pelo método, junto com uma lista contendo quantos frames são utilizados para representar cada palavra.

Para extrair as informações proveniente da câmera RealSense™, é utilizado um objeto do tipo *HandDataExtractor*, que cria uma lista contendo todas as informações das articulações de todos os dedos. Para extrair as informações da imagem RGB, ela é inicialmente convertida para tons de cinza, e um vetor é construído com o tom de cinza normalizado de cada pixel.

Após ter extraído as informações necessárias, elas são formatadas para o formato de leitura do CNTK, e gravadas em um arquivo. A sintaxe necessária pode

ser encontrada no site da Microsoft: https://docs.microsoft.com/en-us/cognitive-toolkit/brainscript-cntktextformat-reader.

O código para gerar a entrada dos dados para a rede neural pode ser encontrado no apêndice D.

4.5.2 Treinamento da Rede

Para o treinamento da rede, vários parâmetros como *learning rate*, *minibatch size*, *epoch size* são definidos, e alguns objetos são construídos, com destaque em dois:

***Trainer*:** Objeto do tipo *Trainer* da biblioteca do CNTK, sua construção é feita passando o modelo da rede a ser treinada, os objetos de critério erro e perda, e o objeto utilizado para o aprendizado da rede. É o responsável por treinar a rede.

***Reader*:** Objeto do tipo *Reader* da biblioteca do CNTK, é responsável por gerenciar a leitura dos dados para serem fornecidos para o treinamento.

Um *loop* é executado N vezes, onde N é a quantia de épocas desejadas, no qual os dados fornecidos pelo *Reader* são utilizados pelo *Trainer* para realizar o treinamento da rede, e o progresso do treinamento da rede é exibido. No final do treinamento, a rede é salva em um arquivo com formato *.dnn*, para ser utilizado posteriormente para análise de dados.

4.6 RESULTADOS

Com a rede neural e suas camadas implementadas, foi executado o código para fazer o treinamento da rede. Os dados treinados foram 281 sinais de LIBRAS divididos entre 13 letras e palavras, sendo elas, ‘A’, ‘B’, ‘C’, ‘D’, ‘F’, ‘L’, ‘I’, ‘O’, ‘R’, ‘U’, ‘V’, ‘Z’ e ‘CASA’. Cada sinal possui uma sequência de *frames* que foram passados para a rede.

Com o arquivo *.dnn* gerado pelo CNTK, foi efetuado o teste com 121 outros sinais das mesmas letras e palavras na qual a rede neural foi treinada e o resultado obtido foi bom para os sinais que não se assemelhavam com outros, ou seja, sinais como, ‘A’ e ‘F’, representados na figura 4.5, que são bem diferentes obtiveram 100% de acerto, entretanto sinais como ‘R’ e ‘V’ que são bem parecidos, obtiveram uma taxa de acerto relativamente baixa.

Figura 4.5 - Alfabeto de LIBRAS

Fonte: Silva; Medeiros, 2010

A tabela 4.1 demonstra os resultados da rede neural quando ela tem um grau de certeza igual ou maior que 93%, ou seja, quando a rede acreditar com 90% de “certeza” que um sinal é, por exemplo, a letra ‘A’, o software não irá mostrar esse retorno.

Tabela 4.1 - Resultados do sofware

Sinais de LIBRAS	13
Threshold	93%
Frames avaliados	2039
Resultados Errados	244
Resultados Corretos	1236

Fonte: autoria própria

A tabela 4.2 indica, dentre aqueles 13 sinais citados acima, quantos frames e quais sinais apresentaram erros dentro do software. Isso demonstra que o sinal ‘Z’ é o sinal que mais obteve respostas erradas para seus frames.

Tabela 4.2 - Resultados com erros

Sinal	Frames Errados
O	1
R	61
Z	182

Fonte: autoria própria

O código para testar a rede neural pode ser encontrado no apêndice F.

5 CONCLUSÃO

Tendo em vista os resultados obtidos e o conhecimento adquirido durante a elaboração deste trabalho, pode-se concluir que a utilização das duas câmeras da Intel®, uma principal capturando dados de profundidade e uma secundária capturando imagem RGB, foi de grande importância para melhorar o reconhecimento de sinais de LIBRAS, devido ao fato de que nem todos os gestos são visíveis pela câmera principal e desses gestos serem capturados pela câmera secundária.

A eficiência da rede neural recorrente utilizando o CNTK também pôde ser comprovada, principalmente pelos erros relativamente baixos encontrados durante a fase de treinamento e avaliação a rede, quanto no teste em tempo real, no qual a avaliação foi rápida e precisa para quase todos os gestos.

Alguns pontos negativos também foram observados, que podem ser corrigidos ou minimizados em trabalhos futuros:

a) Dificuldade em reconhecer gestos similares, como ‘I’ e ‘Z’ ou ‘R’ e ‘V’;
b) Necessidade de um ambiente de avaliação muito similar ao de treino;
c) Baixa confiabilidade dos dados provenientes da câmera de profundidade dependendo da posição das mãos.

REFERÊNCIAS

AZEVEDO, E.; CONCI, A.; LETA, F. R. **Computação gráfica:** teoria e prática**.** 1. ed. Rio de Janeiro: Campus Elsevier, 2007.

BISOL, C. A.; VALENTINI, C. B. **Surdez e Deficiência Auditiva - Qual a diferença?** Objeto de Aprendizagem. UCS/FAPERGS, 2011. Disponível em: <http://www.grupoelri.com.br/Incluir/downloads/OA_SURDEZ_Surdez_X_Def_Audit_Texto.pdf>. Acesso em: 27 de mai. de 2017.

CARDOSO, S. H., Comunicação Entre as Células Nervosas. **Cérebro & Mente**, n. 12, 2000. Disponível em: <http://www.cerebromente.org.br/n12/fundamentos/neurotransmissores/neurotransmitters2_p.html>. Acesso em: 25 ago. 2017.

CARVALHO, A. P. de L. F. **Redes neurais artificiais.** Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>. Acesso em: 25 ago. 2017.

CNTK **Evaluation Overview.** Disponível em: <https://docs.microsoft.com/en-us/cognitive-toolkit/CNTK-Evaluation-Overview>. Acesso em: 26 ago. 2017.

COMPUTER GRAPHICS AND IMAGE PROCESSING, 2, 2008, Campo Grande, **Anais eletrônicos...** Campo Grande: UFMS, 2008. Disponível em: <http://www.gpec.ucdb.br/sibgrapi2008/>. Acesso em: 24 abr. 2017.

DEFICIÊNCIA Auditiva Atinge 9,7 Milhões de Brasileiros. 2013. Disponível em: <http://www.adap.org.br/site/index.php/artigos/20-deficiencia-auditiva-atinge-9-7-milhoes-de-brasileiros>. Acesso em: 19 mar. 2017.

FRIESS, I. I. **Análise de banco de dados NoSQL e desenvolvimento de uma aplicação.** 2013. 100 f. Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Santa Maria, Santa Maria, 2013.

GONÇALVES, L. M. G. **Perceptrons.** 2004. Disponível em: <https://www.dca.ufrn.br/~lmarcos/courses/robotica/notes/perceptrons.pdf>. Acesso em: 25 ago. 2017.

HANDS **On Labs Language Understanding**. Disponível em: <https://github.com/Microsoft/CNTK/wiki/Hands-On-Labs-Language-Understanding>. Acesso em: 19 mar. 2017.

INTEL® **RealSense™ SDK 2016 R2 Documentation**. Disponível em: <https://software.intel.com/sites/landingpage/realsense/camera-sdk/v1.1/documentation/html/index.html?doc_devguide_introduction.html>. Acesso em: 18 mar. 2017.

International Conference on Artificial Neural Networks, 2016, Barcelona. **Proceedings...** Barcelona: Springer, 2016

IYODA, E. M., **Inteligência computacional no projeto automático de redes neurais híbridas e redes neurofuzzy heterogêneas.** 2000. Tese (Mestrado) Faculdade de Engenharia Elétrica e de Computação, Campinas, 2000.

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento digital de imagens.** 1. ed. Rio de Janeiro: Brasport, 1999.

MICROSOFT **Cognitive Toolkit (CNTK).** Disponível em: <https://github.com/Microsoft/CNTK>. Acesso em: 26 ago. 2017.

MORI, N. N. R.; SANDER, R. E. História da Educação dos Surdos no Brasil. In: Seminário de Pesquisa do PPE, 2015, Maringá. **Anais...** Maringá: Universidade Estadual de Maringá, 2015.

NEMES, A. **175 anos de fotografia: conheça a história dessa forma de arte**. Disponível em: <https://www.tecmundo.com.br/fotografia-e-design/60982-175-anos-fotografia-conheca-historia-dessa-forma-arte.htm>. Acesso em: 23 abr. 2017.

PEREIRA, R. L. C. **Reconhecimento mobile de linguagem de LIBRAS.** 2016. 32 f. Dissertação (Iniciação Científica) – Faculdade de Engenharia de Sorocaba, Sorocaba, 2016.

PESSOA, B. C.; et al. Banco de dados MongoDB vs banco de dados SQL Server 2008. **Revista Eletrônica Científica de Ciência da Computação**, Alfenas, n. 1, 2012. Disponível em: <http://revistas.unifenas.br/index.php/RE3C/article/view/18>. Acesso em: 25 mar. 2017.

PHYTON **API for CNTK (2.1)**. 2017. Disponível em: <https://www.cntk.ai/pythondocs/>. Acesso em: 26 ago. 2017.

QUEIROZ, J. E. R., GOMES H. M., Introdução ao Processamento Digital de Imagens. **Revista RITA**, v. 8, n. 2, 2001.

RIOS, L. R. S. **Visão computacional.** Departamento de Ciência da computação - Universidade Federal da Bahia, Salvador, 2010.

SILVA, E. M. da; MEDEIROS, S. M. B. de **Aprendendo a se comunicar usando as mãos**. 2010. Disponível em: <http://portaldoprofessor.mec.gov.br/fichaTecnicaAula.html?aula=24022>. Acesso em: 31 out. 2017.

SILVA, F. I. et al. **Aprendendo língua brasileira de sinais como segunda língua:** nível básico. 1. ed. Palhoça, 2007.

SILVA M.; VELLASCO, M. M. B. R.; CATALDO, E. Modelo híbrido neuro-evolucionário para problemas de agrupamento utilizando redes. In: CONGRESSO BRASILEIRO DE INTELIGÊNCIA COMPUTACIONAL, 2015, Curitiba, PR., **Anais eletrônicos...** Curitiba. Disponível em: <http://abricom.org.br/eventos/cbic_2015/>. Acesso em: 20 mar. 2017.

SONZA, A. P. **Curiosidades Sobre LIBRAS e os Surdos.** Disponível em: <http://blog.acessibilidade.bento.ifrs.edu.br/?p=36>. Acesso em: 29 mar. 2017.

TANENBAU, Andrew Stuart. **Redes de computadores.** 4. ed. Amsterdam: Campos, 2003.

TATIBANA, C. Y.; KAETSU, D. Y., **Redes neurais.** Disponível em: < http://www.din.uem.br/ia/neurais/>. Acesso em: 20 ago. 2017.

THOMÉ, A. G., **Os Modelos Perceptron, Adaline Adaline e Madaline.** 2005. Disponível em:

<http://equipe.nce.ufrj.br/thome/p_grad/nn_ic/transp/T4_perceptron_adaline.pdf>. Acesso em: 26 ago. 2017.

THOMÉ, A. C. G.; MARQUES, A. C. B. **Inteligência Computacional:** Perceptron. 2003. Disponível em: <http://equipe.nce.ufrj.br/thome/grad/nn/mat_didatico/aula7.pdf>. Acesso em: 25 ago. 2017.

VILLEGAS, Alex. **O Controle da cor.** 1. ed. Santa Catarina: Photos, 2009.

YU, D.; YAO, K.; ZHANG, Y. The Computational Network Toolkit. **IEEE Signal Processing Magazine**, v. 32, n. 6, p. 123 – 126, nov. 2015.

ZAMBIASI, S. P., **O neurônio artificial.** Disponível em: <http://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_artificial/index.html>. Acesso em: 20 ago. 2017.

ZUBEN, J. V.; CASTRO L. N., **Redes neurais artificiais.** Disponível em: <ftp://vm1-dca.fee.unicamp.br/pub/docs/vonzuben/ia006_03/topico5_03.pdf>. Acesso em: 20 ago. 2017.

APÊNDICE A – CÓDIGO PARA CAPTURA DE DADOS DA CÂMERA

Abaixo está escrito o código para captura de coordenadas das mãos pelo software:

```
public void Start()
{
   try
   {
      using (PXCMSession session = PXCMSession.CreateInstance())
      {
         using (PXCMSenseManager sm = session.CreateSenseManager())
         {
            base.CheckError(sm.EnableHand());
            base.CheckError(sm.Init());

            PXCMHandModule handModule = this.GetHandModule(sm);

            using (PXCMHandData handData = handModule.CreateOutput())
            {
               while (_isRunning && sm.AcquireFrame(true).IsSuccessful())
               {
                  base.CheckError(handData.Update());

                  ICollection<HandData> data = _handDataService.OnProcess(handData);
                  _dataService.OnProcess(data);
                  sm.ReleaseFrame();
               }
            }
         }
      }
   }
   catch (Exception ex)
   {
      ExceptionHandler exceptionHandler = new ExceptionHandler(ex);
   }
}

private HandData OnProcessHand(PXCMHandData handData,
                               PXCMHandData.AccessOrderType handEnum)
{
   PXCMHandData.IHand ihand = null;
   base.CheckError(handData.QueryHandData(handEnum, 0, out ihand));
   HandData handDataRealsense = null;

   if (ihand != null)
   {
      handDataRealsense = new HandData(ihand, (int)handEnum);
```

```
        if (ihand.HasTrackedJoints())
        {
            handDataRealsense.FingerDatas = _fingerDataService.OnProcessFinger(ihand);
            handDataRealsense.JointDatas = _jointDataService.OnProcessJoint(ihand);
        }
    }
    return handDataRealsense;
}

public ICollection<FingerData> OnProcessFinger(PXCMHandData.IHand ihand)
{
    List<FingerData> fingerDataRealsense = new List<FingerData>();

    for (int i = 0; i < 5; i++)
    {
        PXCMHandData.FingerData fingerData = null;
        base.CheckError(ihand.QueryFingerData((PXCMHandData.FingerType) i,
                                              out fingerData));
        fingerDataRealsense.Add(new FingerData(fingerData, i));
    }
    return fingerDataRealsense;
}


public IDictionary<JointEnum, JointData> OnProcessJoint(PXCMHandData.IHand ihand)
{
IDictionary<JointEnum, JointData> jointDataRealsense =
                                              new Dictionary<JointEnum, JointData>();

    for (int i = 0; i < 22; i++)
    {
        PXCMHandData.JointData jointData = null;
        base.CheckError(ihand.QueryTrackedJoint((PXCMHandData.JointType)i,
                                               out jointData));
        JointData jd = new JointData(jointData, i);
        jointDataRealsense.Add(jd.JointEnum, jd);
    }
    return jointDataRealsense;
}


public JointData(PXCMHandData.JointData jointData, int jointEnum)
{
    JointEnum = (JointEnum)jointEnum;
    Confidence = jointData.confidence;
    JointPositionWorld = new JointPositionWorld(jointData.positionWorld);
    JointPositionImage = new JointPositionImage(jointData.positionImage);
    JointLocalRotation = new JointLocalRotation(jointData.localRotation);
    JointGlobalOrientation = new JointGlobalOrientation(jointData.globalOrientation);
    JointSpeed = new JointSpeed(jointData.speed);
}
```

APÊNDICE B – CÓDIGO PARA CAPTURA DE IMAGENS

Abaixo está escrito o código para captura imagens pelo software:

```
public void Start()
{
   try
   {
      using (PXCMSenseManager sm = PXCMSenseManager.CreateInstance())
      {
         sm.EnableStream(PXCMCapture.StreamType.STREAM_TYPE_COLOR, 640, 480, 30);
         sm.Init();
         while (_isRunning)
         {
            base.CheckError(sm.AcquireFrame(false));
            PXCMCapture.Sample sample = sm.QuerySample();

            if (sample != null && sample.color != null)
            {
               ImageData imageData = null;
               base.CheckError(sample.color.AcquireAccess(Access.ACCESS_READ,
                                                  PXCMImage.PixelFormat.PIXEL_FORMAT_RGB32,
                                                  out imageData));

               sample.color.ReleaseAccess(imageData);
               Bitmap bitmap = imageData.ToBitmap(0, sample.color.info.width,
                                                     sample.color.info.height);

               bitmap = this.ResizeImage(bitmap, 32, 24);
               _dataService.OnProcess(bitmap);
            }
            sm.ReleaseFrame();
         }
      }
   }
   catch (Exception ex)
   {
      ExceptionHandler exceptionHandler = new ExceptionHandler(ex);
   }
}
```

APÊNDICE C – CÓDIGO DE SINCRONIZAÇÃO DOS DADOS DAS CÂMERAS

Abaixo código para sicronização dos dados das cameras:

```
public void Receive(string text, CameraEnum cameraEnum)
{
   DataSocket dataSocket = null;
   try
   {
      dataSocket = SerializeUtil.Deserialize<DataSocket>(text);
      if (dataSocket != null)
      {
         if (dataSocket.RealsenseType == RealsenseTypeEnum.HAND_DATA)
         {
            lock (_dataQueue)
            {
               _dataQueue.Push(dataSocket);
            }
         }
         else if (dataSocket.RealsenseType == RealsenseTypeEnum.IMAGE_DATA)
         {
            lock (_imageQueue)
            {
               _imageQueue.Push(dataSocket);
            }
         }
      }
   }
   catch (Exception ex)
   {
      ExceptionHandler exceptionHandler = new ExceptionHandler(ex);
   }
}


private void ProcessDataSocketQueue(object sender, System.Timers.ElapsedEventArgs e)
{
   while (_dataQueue.Count > 0 && _imageQueue.Count > 0)
   {
      try
      {
         Signal signal = new Signal();
         Task dataTask = Task.Run(() =>
         {
            DataSocket data = null;
            lock (_dataQueue)
            {
               data = _dataQueue.Pop();
            }
```

```
            signal.SetData(data);
            signal.DataFloat = data.CntkInput;
        });
        Task imageTask = Task.Run(() =>
        {
           DataSocket image = null;
           lock (_imageQueue)
           {
              image = _imageQueue.Pop();
           }
           signal.SetImage(image);
           signal.ImageFloat = image.CntkInput;
        });
        dataTask.Wait();
        imageTask.Wait();
        _signalQueue.Enqueue(signal);
     }
     catch (Exception ex)
     {
        ExceptionHandler eh = new ExceptionHandler(ex);
     }
  }
  _timerSocket.Start();
}


private void ProcessSignalQueue(object sender, System.Timers.ElapsedEventArgs e)
{
  while (_signalQueue.Count > 0)
  {
     Signal signal = _signalQueue.Dequeue();
     switch (_controlTypeEnum)
     {
        case ControlTypeEnum.Saving:
           signal.Word = _word;
           signal.FrameNumber = _frameNumber;
           _frameNumber++;
           _signalRepository.Create(signal);
           break;

        case ControlTypeEnum.Default:
           string word = _cntkService.Compute(signal);
           if (!String.IsNullOrWhiteSpace(word))
           {
              _homePage.PlotWord(word);
           }
           break;
      }
   }
   _timerSignal.Start();
}
```

APÊNDICE D – CÓDIGO PARA CRIAR A ENTRADA PARA A REDE NEURAL

Abaixo código que cria o *array* de entrada para a rede neural:

```
public static List<float> Build(Bitmap bitmap)
{
   List<float> list = new List<float>();
   for (int x = 0; x < 32; x++)
   {
      for (int y = 0; y < 24; y++)
      {
         list.Add((float)bitmap.GetPixel(x, y).R / (float)255);
      }
   }

   return list;
}

private static ICollection<float> Build(ICollection<HandData> handData,
                                        HandEnum handEnum)
{
   if (handData == null)
   {
      return new float[340];
   }

   List<float> list = new List<float>();
   HandData hd = null;
   for (int i = 0; i < handData.Count; i++)
   {
      if (handData.ElementAt(i).HandEnum == handEnum)
      {
         hd = handData.ElementAt(i);
         break;
      }
   }

   if (hd == null)
   {
      return new float[340];
   }

   for (int i = 3; i < 23; i++)
   {
      JointData jd = null;
      if (hd.JointDatas.TryGetValue((JointEnum)i, out jd))
      {
         list.AddRange(jd.JointPositionWorld.ToArray());
         list.AddRange(jd.JointPositionImage.ToArray());
```

```
            list.AddRange(jd.JointLocalRotation.ToArray());
            list.AddRange(jd.JointGlobalOrientation.ToArray());
            list.AddRange(jd.JointSpeed.ToArray());
        }
        else
        {
            list.AddRange(new float[17]);
        }
    }

    return list;
}
```

APÊNDICE E – CÓDIGO PARA CRIAR A REDE NEURAL

Abaixo segue código para criar o arquivo .dnn da rede neural:

```
def train(save=False, hidden_dim=521):
    reader=create_reader(path="data/train.txt")
    hand_input, labels = create_containers()
    model = create_model(hidden_dim=hidden_dim,input=hand_input)
    loss, label_error = create_criterion_function(model, labels)

    epoch_size = 100
    lr_per_sample = [3e-4] * 4 + [1.5e-4]
    lr_per_minibatch = [lr * config.MINIBATCH_SIZE for lr in lr_per_sample]
    lr_schedule = C.learning_rate_schedule(lr_per_minibatch, C.UnitType.minibatch,
                                           epoch_size)
    momentum_as_time_constant = C.momentum_as_time_constant_schedule(700)

    learner = C.adam(parameters=model.parameters,
                                  lr=lr_schedule,
                                  momentum=momentum_as_time_constant,
                                  gradient_clipping_threshold_per_sample=15,
                                  gradient_clipping_with_truncation=True)

    progress_printer = C.logging.ProgressPrinter(tag='Training',
                                                 num_epochs=config.MAX_EPOCHS)
    trainer = C.Trainer(model, (loss, label_error), learner, progress_printer)
    C.logging.log_number_of_parameters(model)

    t = 0
    for epoch in range(config.MAX_EPOCHS):
        epoch_end = (epoch + 1) * epoch_size

        while t < epoch_end:
            input_data = reader.next_minibatch(config.MINIBATCH_SIZE, input_map={
                hand_input: reader.streams.data,
                labels: reader.streams.labels
            })

            trainer.train_minibatch(input_data)

            t += input_data[labels].num_samples

        trainer.summarize_training_progress()
   print(trainer.previous_minibatch_evaluation_average)


def create_containers():
    hand_input = C.sequence.input_variable((1, config.DATA_SIZE))
    network_output = C.sequence.input_variable((1, config.NUM_LABELS))
```

```
    return hand_input, network_output

def create_model(hidden_dim=300, input=None):
    h_fwd = Recurrence(LSTM(hidden_dim))(input)
    h_fwd_s = Stabilizer()(h_fwd)
    h_bwd = Recurrence(LSTM(hidden_dim), go_backwards=True)(input)
    h_bwd_s = Stabilizer()(h_bwd)
    h = splice(h_fwd_s, h_bwd_s)
    return Dense(config.NUM_LABELS, name='classify',activation=C.softmax)(h)

def create_reader(path="data/hand_data.txt", is_training=True):
    return C.io.MinibatchSource(C.io.CTFDeserializer(path, C.io.StreamDefs(
        data=C.io.StreamDef(field='data', shape=config.DATA_SIZE),
        labels=C.io.StreamDef(field='label', shape=config.NUM_LABELS)
    )), randomize=is_training, max_sweeps=C.io.INFINITELY_REPEAT if is_training else
1)

def create_criterion_function(model, labels):
    ce = C.cross_entropy_with_softmax(model, labels)
    errs = C.classification_error(model, labels)
    return ce, errs
```

APÊNDICE F – CÓDIGO PARA AVALIAR OS FRAMES

Abaixo código que executa teste da rede neural criada:

```
reader_test = create_reader(path="data/test.txt", is_training=False)
test_input_map = {
    hand_input  : reader_test.streams.data,
    labels  : reader_test.streams.labels,
}

test_minibatch_size = 30
test_result = 0.0
num_minibatches_tested = 0

while True:
    data = reader_test.next_minibatch(test_minibatch_size,
                                      input_map=test_input_map)
    if not data:
        break

    num_minibatches_tested += 1
    eval_error = trainer.test_minibatch(data)
    test_result = test_result + eval_error

error_tax = test_result * 100 / num_minibatches_tested
print("Average test error: {0:.2f}%".format(error_tax))
```