

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO

Fernando Henrique Costa Carneiro

**SISTEMA DE RECONHECIMENTO DE LÍNGUA DE SINAIS USANDO
REDE NEURAL CONVOLUCIONAL E MACHINE LEARNING.**

Apresentação do Projeto:

<https://youtu.be/dOTQca9Yhw4>

Cesário Lange - SP

2022

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO

**SISTEMA DE RECONHECIMENTO DE LÍNGUA DE SINAIS USANDO
REDE NEURAL CONVOLUCIONAL.**

Trabalho de Conclusão de Curso
apresentado ao curso de Engenharia da
Computação da Universidade Virtual do
Estado de São Paulo como requisito
parcial para obtenção do título de
bacharel em Engenharia da Computação.

Orientadora: Dra. Thays de Souza J. Luiz.

Cesário Lange - SP

2022

CARNEIRO, Fernando. **Sistema de Reconhecimento de Língua de Sinais Usando Rede Neural Convolucional.** Relatório Técnico-Científico (Engenharia de Computação) – Universidade Virtual do Estado de São Paulo. Orientadora: Dra. Thays de Souza J. Luiz. Polo: Cesário Lange, 2022.

RESUMO

A proposta deste trabalho é desenvolver um Sistema de Reconhecimento de Língua de Sinais Brasileira, permitindo ao usuário reconhecer certos sinais. As redes neurais convolucionais são um dos algoritmos pertencentes ao aprendizado de máquina e é o método utilizado neste trabalho para alcançar o objetivo de reconhecer uma sinalização. Foi observado durante o projeto a importância de ter um conjunto de dados bem formulado, e que existem diversas formas de parametrizar uma rede neural dependendo do contexto em que ela está inserida. Os resultados foram satisfatórios com uma rede neural convolucional de 4 camadas, com um índice de precisão de mais de 90% de acerto. Esses resultados mostram que essa técnica é bem sucedida em resolver o problema de identificar sinais em vídeo, mesmo que em tempo real, como foi demonstrado.

PALAVRAS-CHAVE: Linguagem De Sinais (LIBRAS), Reconhecimento De Imagem, Aprendizado De Máquina, Rede Neural Convolucional.

CARNEIRO, Fernando. **Sign Language Recognition Using Convolutional Neural Network and Machine Learning.** Technical-Scientific Report (Computer Engineering) - Virtual University of the State of São Paulo. Advisor: Dr. Thays de Souza J. Luiz. Pole: Cesário Lange, 2022.

ABSTRACT

The purpose of this work is to develop a Brazilian Sign Language Recognition System, allowing the user to recognize certain signs. Convolutional neural networks are one of the algorithms belonging to machine learning and it is the method used in this work to achieve the goal of recognizing a signal. It was observed during the project the importance of having a well-formulated data set, and that there are several ways to parameterize a neural network depending on the context in which it is inserted. The results were satisfactory with a 4-layer convolutional neural network, with an accuracy rate of more than 90% of success. These results show that this technique is successful in solving the problem of identifying video signals, even in real time, as demonstrated.

KEYWORDS: Brazilian Sign Language, Image Recognition, Machine Learning, Convolutional Neural Network.

TABELA DE FIGURAS

Figura 1	- Problema de classificação de e-mails	6
Figura 2	- Exemplo de uma rede feedforward	9
Figura 3	- Representação de uma ativação ReLU.	11
Figura 4	- Interface do aplicativo Spyder	15
Figura 5	- Algoritmo para captura de imagens	16
Figura 6	- Exemplo de imagens capturadas	17
Figura 7	- Inicialização de uma rede neural	18
Figura 8	- Camadas de convolução da rede	19
Figura 9	- Camada totalmente conectada	20
Figura 10	- Preparação dos dados	21
Figura 11	- Parâmetros de treinamento	22
Figura 12	- Carregando os modelos gerados	23
Figura 13	- Bloco de código em laço de repetição	24
Figura 14	- Inicialização da captura de vídeo	25
Figura 15	- Criando zona de interesse	25
Figura 16	- Função de previsão	26
Figura 17	- Exibição do resultado	27
Figura 18	- Função para criação de gráficos.	28
Figura 19	- Dataset com Image Augmentation	29
Figura 20	- Comparação de amostra suja e limpa	31
Figura 21	- Erros aos adicionar quinta camada.	34
Figura 22	- Reconhecimento em tempo real	35

TABELA DE GRÁFICOS

Gráfico 1 - Resultados com Data Augmentation	29
Gráfico 2 - Resultados sem Data Augmentation	30
Gráfico 3 - Treinamento com 2 camadas e dataset sujo.	30
Gráfico 4 - Treinamento com 3 camadas e dataset sujo	31
Gráfico 5 - Treinamento com 2 camadas e dataset limpo	32
Gráfico 6 - Treinamento com 3 camadas e dataset limpo	32
Gráfico 7 - Treinamento com 4 camadas	33

SUMÁRIO

1. INTRODUÇÃO	1
2. LEVANTAMENTO E DELIMITAÇÃO DO PROBLEMA (TEMA)	1
2.1 Problema	1
2.2 Delimitação	2
3. JUSTIFICATIVA	2
4. OBJETIVOS	2
4.1 Objetivo Geral	2
4.2 Objetivos Específicos	3
5. FUNDAMENTAÇÃO TEÓRICA	3
5.1 Inteligência	3
5.2 Inteligência Artificial	3
5.2.1 Início histórico	3
5.2.2 O Teste de Turing	4
5.2.3 Aplicações da IA	4
5.3 Aprendizado de máquina (machine learning)	5
5.3.1 Aprendizado Supervisionado	6
5.3.2 Aprendizado Não-Supervisionado	7
5.3.3 Aprendizado Semi-Supervisionado	7
5.3.4 Aprendizado por Reforço	8
5.4 Redes Neurais Artificiais	8
5.4.1 Equações	9
5.4.2 Função de ativação ReLU	11
5.4.3 Backpropagation	12
5.5 Redes Neurais Convolucionais	12
5.5.1 Camada de Convolução	12
5.5.2 Camada de Pooling	12
5.5.2 Camada de Totalmente Conectada	13
6. METODOLOGIA	13
6.1 Conjunto de dados (dataset)	13
7. DESENVOLVIMENTO	14
7.1 Conjunto de dados (dataset)	14
7.2 Algoritmo de aprendizagem	17
7.2.1 Construção da rede neural	17
7.2.2 Preparação dos dados para treino	20
7.2.3 Parâmetros de treinamento	21
7.3 Algoritmo de reconhecimento	22

7.3.1 Carregando os modelos gerados	22
7.3.2 Captura de vídeo para o reconhecimento	23
7.3.3 Definindo uma zona de interesse	25
7.3.4 Reconhecimento com Inteligência Artificial	25
7.3.5 Exibição do resultado	26
8. RESULTADOS	27
8.1 Algoritmo de renderização de gráfico	27
8.2 Modificações no dataset (Data Augmentation)	28
8.3 Limpeza do dataset	30
8.4 Camadas de convolução	33
8.5 Reconhecimento em tempo real	35
9. CONSIDERAÇÕES FINAIS	36
REFERÊNCIAS	37

1. INTRODUÇÃO

A língua de sinais é uma forma não verbal de comunicação empregada para expressar sentimentos, pensamentos e emoções usada por pessoas com deficiência auditiva e de fala. No Brasil existem apenas duas línguas oficiais, o Português e a Língua Brasileira de Sinais (Libras), reconhecida oficialmente em 2002 pela Lei nº 10.436 (BRASIL, 2002). Intérpretes de linguagem de sinais têm sido cada vez mais procurados e são necessários durante consultas médicas e legais, sessões educacionais e de treinamento. Um sistema automatizado de reconhecimento usando inteligência artificial reduziria o impacto do déficit de profissionais intérpretes de LIBRAS (Línguagem Brasileira de Sinais). O projeto que propomos usa uma CNN (*Convolutional Neural Network*) em duas dimensões para extrair os recursos necessários para o reconhecimento do vídeo e posterior identificação.

2. LEVANTAMENTO E DELIMITAÇÃO DO PROBLEMA

2.1 Problema

De acordo com o IBGE (2010) uma a cada vinte pessoas no Brasil é surda, são aproximadamente 10 milhões de pessoas com deficiência auditiva. É um número expressivo, e o Brasil não é exceção, a Organização Mundial da Saúde estima que hoje haja mais de 1,5 bilhão de pessoas com perda auditiva (WHO, 2021b) e esse número pode aumentar para 2,5 bilhão de pessoas até 2050 (WHO, 2021c). O impacto vai além do âmbito relacionado à saúde e atinge de modo significativo toda a economia global, ainda segundo um relatório da Organização Mundial da Saúde:

A OMS estima que a perda auditiva não tratada representa um custo global anual de US\$ 980 bilhões de dólares. Isso inclui custos do setor de saúde (excluindo o custo de aparelhos auditivos), custos de apoio educacional, perda de produtividade e custos sociais. 57% desses custos são atribuídos a países de baixa e média renda. (WHO, 2021a, p.1)

Sem dúvida investimentos de quase 1 trilhão de dólares são significativos, considerando a tendência do aumento de pessoas que dependem desses recursos podemos naturalmente esperar o aumento da necessidade de investimentos também.

2.2 Delimitação

Apesar dos investimentos feitos, ainda há carência em uma das principais necessidades da sociedade surda, que é a prestação de serviços de interpretação de língua de sinais, atividade essa regulamentada em 2010 no Brasil (2010). Nos delimitaremos justamente a atuar nessa causa, provendo uma opção ao intérprete humano e parcialmente suprindo essa necessidade emergente.

3. JUSTIFICATIVA

Uma das soluções prováveis nesse âmbito é o objetivo do projeto proposto que visa capturar uma imagem em língua de sinais, reconhecer o movimento e interpretá-lo para o Português escrito. Um sistema automatizado de reconhecimento e tradução de LIBRAS usando inteligência artificial seria útil em diversas situações, veja por exemplo duas delas:

- Aprendizado de LIBRAS: seria um grande auxílio para quem está aprendendo a língua na identificação dos sinais.
- Atendimento ao deficiente auditivo: tornaria possível a compreensão da sinalização de um surdo que procura atendimento em um estabelecimento por uma pessoa não fluente em LIBRAS, por exemplo: em uma loja, departamento público ou hospital.

4. OBJETIVOS

4.1 Objetivo Geral

O trabalho a seguir tem por objetivo a criação de um sistema para o reconhecimento de sinalização em Língua Brasileira de Sinais, classificação e tradução para a língua Portuguesa, aplicando os conhecimentos mais atuais em algoritmos de redes neurais e machine learning.

4.2 Objetivos Específicos

- Levantar os conhecimentos mais atuais em Inteligência Artificial e Machine Learning;
- Entender e validar o treinamento de uma rede neural convolucional;
- Fazer avaliação do uso da tecnologia escolhida para alcançar os objetivos do sistema de reconhecimento e tradução de língua de sinais;

5. FUNDAMENTAÇÃO TEÓRICA

5.1 Inteligência

A definição de inteligência é um tanto controversa, no entanto, de maneira geral uma sugestão bem aceita é a de Gottfredson (1997):

Envolve a capacidade de raciocinar, planejar, resolver problemas, pensar abstratamente, compreender ideias complexas, aprender rapidamente e aprender com a experiência (...) reflete uma capacidade mais ampla e profunda de compreender o que nos cerca — "perceber", "dar sentido" às coisas ou "descobrir" o que fazer. (GOTTFREDSON, 1997, p.13)

Mas basta um pouco de pesquisa para encontrar diferentes pontos de vista sobre a definição de inteligência, por exemplo, alguns estudiosos levam em consideração como a inteligência não é consistente e varia dependendo de circunstâncias, domínios e dos critérios de julgamento. (NEISSER, et al, 1996).

A inteligência artificial é diferente da programação habitual por ter a capacidade de lidar com a incerteza e constante variabilidade encontrada no contexto do mundo real e mesmo assim ser bem sucedida, usando conhecimento e experiência obtidos, para chegar a conclusões e alcançar objetivos.

5.2 Inteligência Artificial

5.2.1 Início histórico

Segundo o autor Bringsjord (2018), o termo inteligência artificial teve seu aparecimento em 1956, durante a conferência de verão patrocinada pela DARPA no Dartmouth College, em Hanover, New Hampshire, (onde dez pensadores se reuniram para o que foi essencialmente um grande brainstorm que durou algumas semanas (DARTMOUTH WORKSHOP, 2022). No entanto, o campo em si já existia mesmo antes do seu nome como conhecemos hoje. Alguns anos antes, o matemático inglês Alan Turing argumentara em um artigo, hoje muito famoso, chamado Computing Machinery and Intelligence (Turing, 1950): “Pode uma máquina

pensar?" Neste mesmo artigo foi proposto um por ele um jogo chamado "Jogo da Imitação".

5.2.2 O Teste de Turing

O jogo da imitação foi uma introdução para o até hoje conhecido Teste de Turing, o jogo consistia no seguinte:

É jogado com três pessoas, um homem (A), uma mulher (B) e um interrogador (C) que podem ser de ambos os sexos. O interrogador fica em uma sala separada dos outros dois. O objetivo do jogo para o interrogador é determinar qual dos outros dois é o homem e qual é a mulher... O objetivo do jogo para o terceiro jogador (B) é ajudar o interrogador. (Turing, 1950)

Seu intuito era averiguar se a performance linguística de um indivíduo era suficiente para enganar o interrogador. Com esse contexto, Turing propôs que o papel do jogador A fosse feito por um computador, ou seja, fazer o papel do homem a levar o interrogador ao erro e que se esse fosse bem sucedido no teste então poderia se dizer que a resposta para a pergunta: "pode uma máquina pensar", é sim.

A relevância do Teste de Turing para a área de inteligência artificial é devido a nenhuma das diversas áreas do conhecimento envolvidas, como filosofia, psicologia e neurociência, terem sucesso em conseguir definir plenamente "inteligência" e "pensamento", não o suficiente para modelar a inteligência em uma máquina. O teste de Turing em sua simplicidade ao menos provê algo que pode ser medido, uma solução pragmática para uma questão filosófica ainda difícil.

5.2.3 Aplicações da IA

Atualmente o campo da inteligência artificial está em alta, e o motivo é que estamos começando a ver suas aplicações práticas, se antes já havia empolgação entre os estudiosos com os avanços teóricos, hoje temos os recursos necessários para ver essa empolgante realidade.

Uma das aplicações mais usadas hoje em dia é a de tradutores automáticos, essa área é conhecida como Processamento de Linguagem Natural (SETHUNYA et al., 2016) e parte do seu sucesso é devido a grande quantidade de dados em texto disponíveis para o treinamento dos modelos de inteligência artificial. O Google

tradutor é um ótimo exemplo, com suporte a mais de 100 línguas ele reconhece e traduz tanto de forma escrita como falada. (GOOGLE, 2022)

A maioria de nós já ouviu falar dos carros autônomos, empresas como Spark.ai, Quantum Computing e Ephel estão próximas de criar um sistema autônomo ideal, isso porque atualmente os veículos autônomos são equipados com uma série de câmeras, radares e sensores LiDAR (detecção e alcance de luz), que lhes permitem navegar em condições que seriam complicadas para os humanos, como dirigir em um banco de neblina ou navegar em um breu. (DELOHERY, 2022). Do outro lado, empresas multinacionais consolidadas estão investindo pesado como a alemã Volkswagen que anunciou recentemente investimentos de mais de 2 bilhões de dólares no setor de mobilidade autônoma. (HOPE, 2022)

Outra área em que há otimismo em relação aos resultados é o de diagnósticos automáticos. Diagnósticos computadorizados usando inteligência artificial já têm demonstrado resultados melhores do que o feito por humanos. A empresa iFlytek criou o robô Xiaoyi, em 2017 passou no exame para licenciamento de médicos da China (Saracco, 2017). O “robô médico” registra os sintomas dos pacientes, analisa as imagens de tomografia computadorizada e faz o diagnóstico inicial.

Esses são apenas algumas das aplicações da inteligência artificial, ainda temos muitas outras como o uso no mercado financeiro, os sistemas de recomendação de filmes, prevenção de ataques cibernéticos, etc.

5.3 Aprendizado de máquina (*machine learning*)

A inteligência artificial não é programável, como um programa de computador em que se cria um algoritmo explicitando o que a máquina deve fazer, mas ao invés, ela é adquirida a partir do recebimento de muitos dados amostrais e da aplicação de métodos de aprendizagem que extraem *insights* e fazem previsões, esse processo é chamado de aprendizado de máquina (*machine learning*).

Segundo o cientista da computação Tom Mitchell, aprendizado de máquina pode ser definido pelo seguinte:

Definição: Diz-se que um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho em tarefas em T, medido por P, melhora com a experiência E. (MITCHELL, 1997, p. 2).

O aprendizado de máquina é progressivo, geram hipóteses a partir do recebimento dos dados, quanto maior a quantidade, melhor. É importante também se ater à qualidade dos dados, pois reflete diretamente na chance do aprendizado ser bem sucedido e entregar os resultados esperados.

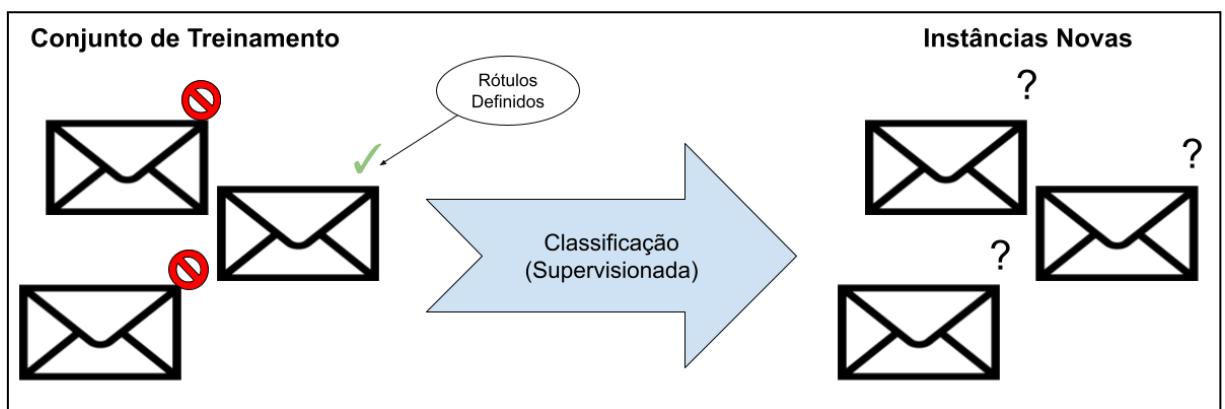
Existem quatro categorias principais de aprendizado: Supervisionado, Semi-Supervisionado, Não Supervisionado e por Reforço. (GÉRON , 2021, p. 8)

5.3.1 Aprendizado Supervisionado

No aprendizado supervisionado os dados do conjunto de treinamento vem com etiquetas ou rótulos que indicam as soluções desejadas. Geralmente esse tipo de aprendizado é usado em tarefas de classificação, como por exemplo classificar se um e-mail é SPAM.

Na figura 1 é possível ver que as instâncias dentro do conjunto de treinamento estão rotuladas, essas informações são inseridas por um supervisor (humano) e por isso o nome supervisionado, com elas o algoritmo consegue classificar as novas instâncias comparando-as ao conhecimento obtido do treinamento.

Figura 1 - Problema de classificação de e-mails



Fonte: Autor

Existem outros problemas passíveis de solução mediante o treinamento supervisionado, geralmente problemas de regressão onde é possível prever valores com base em dados anteriores. Alguns dos algoritmos mais utilizados nesta forma de aprendizado são:

- K-ésimo vizinho mais próximo
- Regressão linear
- Regressão logística
- Máquinas de vetores de suporte
- Árvores de decisão e florestas aleatórias
- Redes Neurais (alguns casos)

5.3.2 Aprendizado Não-Supervisionado

No aprendizado não-supervisionado os dados de treinamento não recebem rótulos, o sistema tenta defini-los sozinho por perceber diferenças entre as instâncias e agrupá-las por similaridade.

Além de seu uso para o agrupamento de instâncias similares, também chamado de clusterização, ele pode ser usado para a detecção de anomalias, tarefa essa que é amplamente utilizada para identificação de fraudes ou defeitos de fabricação.

Alguns dos algoritmos mais utilizados nesta forma de aprendizado são:

- K-Means (Clusterização K-Média)
- Análise de Cluster Hierárquica
- Floresta de Isolamento
- Análise de Componentes Principais
- Apriori

5.3.3 Aprendizado Semi-Supervisionado

Este aprendizado contém uma base de dados de treinamento parcialmente rotulada, ou seja, algumas poucas instâncias rotuladas e muitas não rotuladas. Muitas vezes essa forma de aprendizado consiste em parte supervisionada e parte não-supervisionada como acontece por exemplo no serviço de hospedagem de fotos Google Fotos. O algoritmo reconhecerá automaticamente que há diferentes pessoas nas fotos, essa é a clusterização ou a parte não-supervisionada, no entanto se em apenas uma das fotos for adicionado o nome da pessoa, o sistema usará este rótulo para identificar a mesma em todas as outras fotos, essa seria a classificação ou a parte supervisionada. (GÉRON , 2021, p. 12)

5.3.4 Aprendizado por Reforço

No aprendizado por reforço as instâncias recebem um feedback em forma de sinais, recompensa ou punição, toda vez que confrontado com o problema ele gera uma hipótese baseado nos exemplos passados e verifica o resultado obtido e qual o sinal de reforço deve ser gerado.

Esse tipo de aprendizado é muito usado em jogos, onde o computador aprende jogando contra um oponente humano. Um exemplo é o programa AlphaGo, desenvolvido pela DeepMind Technologies, que joga Go e em 2016 derrotou o jogador profissional Lee Sedol e em 2017 derrotou Ke Jie o melhor jogador do mundo na época. (DEEPMIND, 2022).

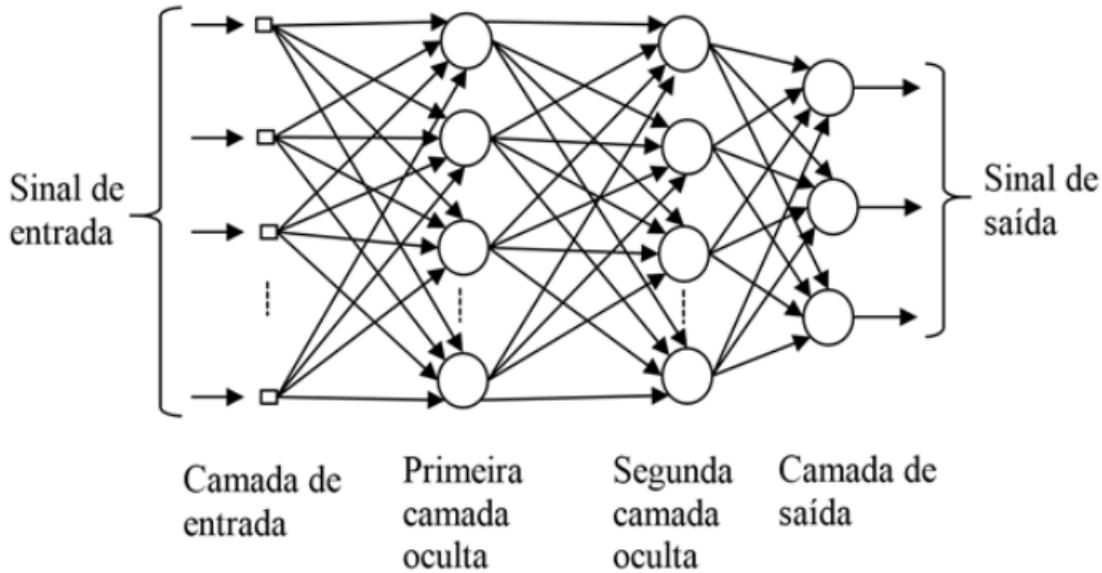
Uma das técnicas de Aprendizado de Máquina que tem tido sucesso em resolver muitos problemas são as Redes Neurais Artificiais (RNA).

5.4 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) são modelos computacionais que se inspiram nas estruturas neurais biológicas dos animais (cérebro principalmente) e que têm a capacidade de realizar aprendizado de máquina bem como o reconhecimento de padrões. Os neurônios artificiais foram propostos pela primeira vez em 1943 por MCCULLOCH e PITTS (1943), mas foi em 1958 que o psicólogo americano Frank Rosenblatt implementou esses estudos em um algoritmo chamado Perceptron, o primeiro algoritmo de aprendizado usando redes neurais (ROSENBLAT, 1957). Hoje usamos múltiplas camadas em uma rede neural, por isso usamos o nome *multi-layered perceptron (MLP)* para redes neurais com até 3 camadas e *Deep Learning* quando usadas redes neurais com mais camadas. (EDA, 2020).

Uma rede neural consiste em muitos neurônios, cada neurônio tem funções definidas, são elas: somar as suas entradas, multiplicá-las por valores de pesos, acrescentar um viés e aplicar uma função degrau como por exemplo a função *Heaviside*.

Figura 2 - Exemplo de uma rede neural



Fonte: Nascimento & Oliveira (2016)

As entradas são alimentadas por sensores, é onde entram os dados no algoritmo, por exemplo, o valor de cor de um pixel. Vemos na figura 2 uma representação de uma rede neural com 3 camadas.

5.4.1 Equações

A propagação dos dados dentro da rede são feitos através de funções de álgebra linear. Antes de demonstrarmos precisamos entender o conceito de vetor e matriz. Um vetor é uma entidade no espaço com valor em quantidade que tem magnitude e direção, sua existência é denotada por:

Notação 1 - Notação de vetor no espaço real

$$x \in \mathbb{R}^2$$

Fonte: Autor

Uma matriz é uma lista de valores bidimensionais composta por duas colunas de arrays. Sua representação no espaço é dada por:

Notação 2 - Matriz bidimensional com 3 linhas e 2 colunas

$$W \in \mathbb{R}^{3*2}$$

Fonte: Autor

No contexto de aprendizado de máquina o vetor é um *array*, ou conjunto, dinâmico de dados que geralmente representam características. O uso de arrays e matrizes é ideal devido a possibilidade de paralelização, onde ao invés de os cálculos serem feitos sequencialmente em uma fila, eles são feitos simultaneamente, resultando num ganho extraordinário de velocidade (VIJAY, 2018).

Na equação 1, vemos os pesos (w) que é uma matriz de valores multiplicados por um vetor de sinais de entrada (x) que definirão a influência do sinal e ajudarão a determinar se aquele neurônio será ativado ou não. O viés ou *bias* (b) é uma constante definida com a atribuição de ajustar melhor o modelo ao conjunto de dados. E uma função de degrau (f) irá transformar os valores numéricos dos sinais possibilitando a diferenciação clara entre ativação e desativação, como veremos adiante.

Equação 1 - Calculando saída de uma camada

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = f \left(\left(\begin{bmatrix} w1 & w2 \\ w3 & w4 \\ w5 & w6 \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \end{bmatrix} \right) + b \right)$$

Fonte: Autor

h = representa a próxima camada oculta (*hidden layer*)

x = representa a entrada de dados

W = representa uma matriz de pesos (*weights*)

b = representa o viés (*bias*)

f = representa a função degrau

Como já dito anteriormente o viés irá fazer um ajuste fino entre o algoritmo e os dados do mundo real e a função degrau irá determinar a ativação do sinal, Antigamente era muito usada a função de *Heaviside* que é nula para valor negativo

e vale 1 (ativação) para um valor positivo (UFRGS - UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2022). No entanto, hoje se tem usado na maioria dos modelos a função linear retificada (*rectified linear unit*) ou simplesmente *ReLU*.

5.4.2 Função de ativação ReLU

A função ReLU tem como resultado para entradas negativas o valor 0 e para entradas positivas o próprio valor de entrada, essa característica tende a diminuir a carga computacional por desativar boa parte dos neurônios durante a propagação. Ela é definida e representada pela equação 2:

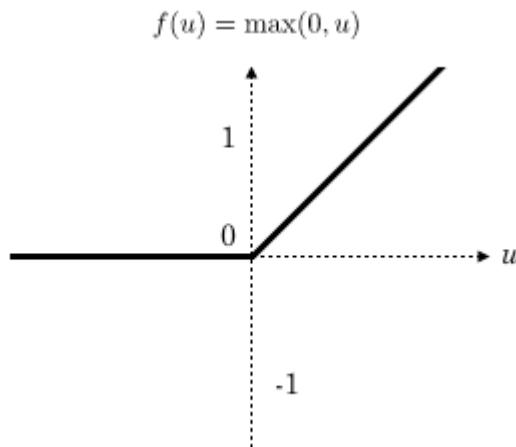
Equação 2 - Função de ativação ReLU

$$\text{ReLU}(x) = \max(0, x)$$

Fonte: Autor

Como visto na equação acima, ela é simples pois só exige que seja computado uma função max(), diferente de outras funções de ativação como sigmóide e *tanh* que trabalham com cálculo de expoentes, o que sobrecarrega o processo.

Figura 3: Representação de uma ativação ReLU



Fonte: Pauly A et al. (2017)

5.4.3 Backpropagation

As camadas da rede após a propagação dos dados, recebem um feedback do resultado, se correto ou não, chamado *backpropagation* que ajusta os pesos de entrada (w) para diminuir a margem de erro até o mínimo possível (CARVALHO, 2009). Desta forma junto de alguma função de erro, como a função de erro quadrático médio (KAY, 1998), a rede é treinada para encontrar padrões, e quando com os valores de peso ideais consegue inferir corretamente em entradas de dados novos. (BARCA et al, 2005).

5.5 Redes Neurais Convolucionais

As Redes Neurais Artificiais (Convolutional Neural Networks), ou simplesmente RNAs, foram primeiramente propostas pelo pesquisador francês Yann LeCun (LECUN et al, 1998). O nome “rede neural convolucional” indica que a rede se utiliza da operação matemática chamada convolução, que é um tipo especializado de operação linear. Redes convolucionais são simplesmente redes neurais que usam convolução no lugar da multiplicação geral de matrizes em pelo menos uma de suas camadas. Se provou ser uma rede muito eficaz de maneira prática, ela é composta geralmente por 3 camadas: convolução, pooling e rede totalmente conectada.

5.5.1 Camada de Convolução

Esta camada da rede tem por objetivo a extração de características da entrada de dados, isso se dá por meio da manipulação dos dados com filtros convolucionais que convoluem as entradas e passam adiante uma simplificação, ou mapa de características, daquela imagem. Para isso os dados são recortados em tamanhos menores e percorridos pelo filtro que ao varrer os dados dispara ao reconhecer determinados padrões como linhas, arestas, cores, etc. Quanto maior a quantidade de camadas mais específica se torna a extração dessas características.

5.5.2 Camada de Pooling

Essa camada substitui a saída da rede em um determinado local por uma estatística resumida das saídas próximas. Por exemplo, a operação de *max pooling* (ZHOU; CHELLAPPA, 1988, p. 71) retorna os elementos máximos de uma área,

mantendo a proeminência daquele grupo de dados. Esse processo diminui a quantidade de dados aumentando a eficiência na computação.

5.5.2 Camada de Totalmente Conectada

A camada totalmente conectada é como o nome diz a conexão entre todas as saídas ativas dos processos anteriores, é a camada final que classifica as características extraídas. As camadas finais já conseguem extrair características mais complexas como dito anteriormente, nessa camada serão correlacionadas às características com a classe do objeto, por exemplo: uma imagem de um gato dispararia a ativação das características correspondentes a rabo, focinho, bigode.

6. METODOLOGIA

Este trabalho consiste primeiramente no decorrimento sob a história e princípios acerca do tópico Inteligência Artificial. Para isso, uma ampla pesquisa bibliográfica foi feita para compreender os complexos mecanismos envolvidos em tal tecnologia, que envolve diversos conceitos como Matemática, Neurociência, Estatística e Física.

Após o fundamento teórico, foram aplicados os conceitos apresentados no desenvolvimento de um algoritmo de *machine learning* capaz de aprender e reconhecer sinalizações em língua brasileira de sinais.

6.1 Conjunto de dados (dataset)

Para o treinamento do algoritmo de Inteligência Artificial, foi desenvolvido um conjunto de dados, ou *dataset*, com a captura de imagens com as letras U, N, I, V, E, S e P sinalizadas em Língua de Sinais. Foi usado um sinalizador, o próprio autor do projeto, a quantidade de capturas foi de 150 capturas para cada uma das 7 letras sinalizadas.

A captura foi feita através de um script simples, na linguagem *Python*, que faz o acesso à webcam e salva a captura de acordo com a letra pressionada no teclado. Ele foi adaptado baseado em um algoritmo, sob autoria de Adarsh Menon (2018), compartilhado sem licença definida no site GitHub.

6.2 Plataforma de desenvolvimento

Para o desenvolvimento do modelo de machine learning foi utilizado a linguagem de programação *Python* que é a linguagem padrão utilizada. Para a captura de imagens foi usado um ambiente de desenvolvimento *Python* chamado *Spyder*. A plataforma de programação de código aberto *Jupyter Notebook* foi a principal ferramenta no desenvolvimento do algoritmo, ela funciona na maioria das plataformas computacionais, uma de suas principais vantagens é a escrita de algoritmos claros, organizados em blocos.

6.3 Bibliotecas de código aberto

Foram usadas algumas bibliotecas de código aberto para o desenvolvimento do projeto. Bibliotecas no contexto de engenharia de programação são conjuntos de sub-programas com uma função específica, prontos para serem utilizados dentro de uma aplicação. As bibliotecas de código aberto, são bibliotecas que têm seu código fonte aberto ao público para consulta e/ou modificação, muitas delas são geridas por grandes entidades ou grupos de pesquisadores. Neste projeto foram usadas as seguintes bibliotecas: *Tensorflow*, *Keras* e *OpenCV*.

A biblioteca *Tensorflow* foi desenvolvida inicialmente pelo Google, seu objetivo é a criação, treinamento e inferência de aprendizado de máquina profundo. A biblioteca *Keras* funciona em cima do *Tensorflow* e visa simplificar o desenvolvimento de redes neurais artificiais, ela conta com diversas ferramentas para implementação de métodos científicos complexos. Desenvolvida inicialmente pela *Intel*, a biblioteca *OpenCV* auxilia no desenvolvimento de sistemas de visão computacional, suas ferramentas proporcionam o uso de imagem e vídeo em tempo real em modelos de machine learning.

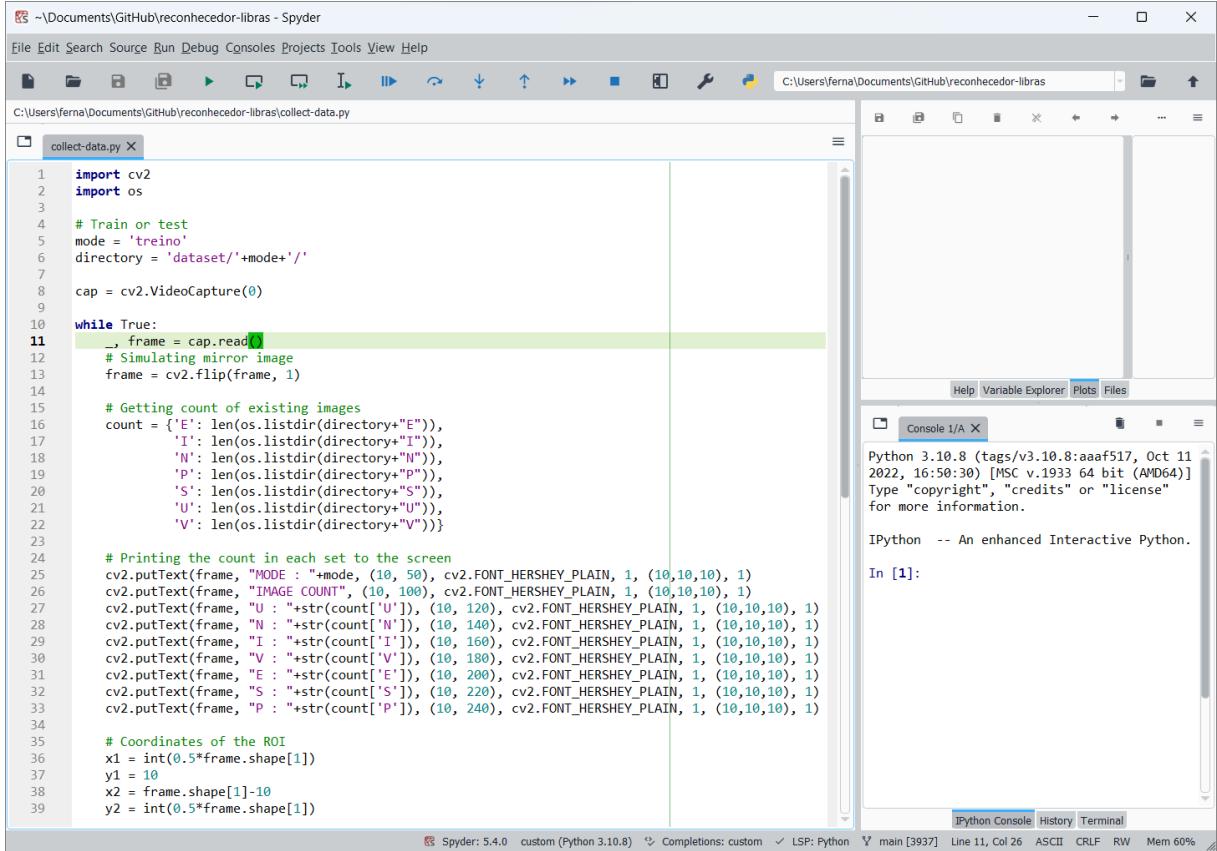
7. DESENVOLVIMENTO

7.1 Conjunto de dados (dataset)

O ambiente de desenvolvimento *Spyder* é muito completo e conta com um editor com funções como realce da sintaxe e de autocompletar o código automaticamente, além de um terminal de comandos e um console para acompanhar o funcionamento.

Abaixo na figura 4, é mostrada uma captura de tela da interface do aplicativo *Spyder*:

Figura 4: Interface do aplicativo *Spyder*



Fonte: Autor

O código fonte do algoritmo foi todo reformulado às necessidades do projeto e desenvolvido dentro do ambiente *Spyder*. Vemos na captura de tela da figura 5, o algoritmo desenvolvido:

Figura 5: Algoritmo para captura de imagens

```

import cv2
import os

# Train or test
mode = 'treino'
directory = 'dataset/'+mode+'/'

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {'E': len(os.listdir(directory+"E")),
              'I': len(os.listdir(directory+"I")),
              'N': len(os.listdir(directory+"N")),
              'P': len(os.listdir(directory+"P")),
              'S': len(os.listdir(directory+"S")),
              'U': len(os.listdir(directory+"U")),
              'V': len(os.listdir(directory+"V"))}

    # Printing the count in each set to the screen
    cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "IMAGE COUNT", (10, 100), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "U : "+str(count['U']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "N : "+str(count['N']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "I : "+str(count['I']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "V : "+str(count['V']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "E : "+str(count['E']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "S : "+str(count['S']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)
    cv2.putText(frame, "P : "+str(count['P']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (10,10,10), 1)

    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])

    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)

    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]
    roi = cv2.resize(roi, (64, 64))

    cv2.imshow("Frame", frame)

    # do the processing after capturing the image!
    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,2))
    roi = cv2.morphologyEx(roi, cv2.MORPH_OPEN, kernel,iterations=1)
    #_, roi = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
    cv2.imshow("ROI", roi)

    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27: # esc key
        break
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory+'U/'+str(count['U'])+'.png', roi)
    if interrupt & 0xFF == ord('n'):
        cv2.imwrite(directory+'N/'+str(count['N'])+'.png', roi)
    if interrupt & 0xFF == ord('i'):
        cv2.imwrite(directory+'I/'+str(count['I'])+'.png', roi)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory+'V/'+str(count['V'])+'.png', roi)
    if interrupt & 0xFF == ord('e'):
        cv2.imwrite(directory+'E/'+str(count['E'])+'.png', roi)
    if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory+'S/'+str(count['S'])+'.png', roi)
    if interrupt & 0xFF == ord('p'):
        cv2.imwrite(directory+'P/'+str(count['P'])+'.png', roi)

cap.release()
cv2.destroyAllWindows()

```

Fonte: Autor

A configuração da imagem capturada foi definida da seguinte forma: dimensões de 64 pixels de largura por 64 pixels de altura (64x64), resolução de 96 dpi (*dots per inch*), intensidade de 8 bits e 1 canal de coloração, ou seja, em escala de cinzas. Segue na figura 6 abaixo, uma amostra.

Figura 6: Exemplo de imagens capturadas



Fonte: Autor

7.2 Algoritmo de aprendizagem

Durante o projeto foi desenvolvido um algoritmo de aprendizagem com redes neurais convolucionais usando os princípios apresentados durante a fundamentação. O algoritmo foi desenvolvido em *Python* e com auxílio das bibliotecas mencionadas anteriormente.

7.2.1 Construção da rede neural

A construção da rede neural convolucional consiste nos seguintes passos gerais: convolução, pooling e camada totalmente conectada. Durante eles, alguns outros passos são adicionados para aperfeiçoar o resultado.

Na figura 7, é possível ver a construção da rede neural, primeiramente é inicializada usando a biblioteca *Keras* e o método *Sequential()*.

Figura 7: Inicialização de uma rede neural

Importando as bibliotecas necessárias

```
In [2]: from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.preprocessing.image import ImageDataGenerator
import os
```

Construindo a rede neural

Inicialização

```
In [3]: model = Sequential()
```

Fonte: Autor

Após isso moldamos nossa rede por adicionar as camadas, como vemos abaixo na figura 8, começamos pela primeira camada de convolução, no projeto foram usadas quatro camadas de convolução de duas dimensões (2D) com tamanho de três pixels de altura e largura (3x3) e ativação ReLU. O parâmetro *input_shape* comunica o tamanho da imagem de entrada e o parâmetro *strides* determina o tamanho do passo de deslocamento *pooling*.

Figura 8: Camadas de convolução da rede

Primeira camada de convolução e pooling

```
In [4]: model.add(Convolution2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))  
In [5]: model.add(MaxPooling2D(pool_size=(2, 2), strides=2))  
In [6]: model.add(Dropout(0.25))
```

Segunda camada de convolução e pooling

```
In [7]: model.add(Convolution2D(32, (3, 3), activation='relu'))  
In [8]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

Terceira camada de convolução e pooling

```
In [9]: model.add(Convolution2D(64, (3, 3), activation='relu'))  
In [10]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

Quarta camada de convolução e pooling

```
In [11]: model.add(Convolution2D(64, (3, 3), activation='relu'))  
In [12]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

Fonte: Autor

É possível ainda notar naa figura 8, que o primeiro parâmetro da função de convolução tem seu valor diferente em metade das camadas. Esse parâmetro se chama filtro e determina a quantidade de *kernels* (O termo *kernel* se refere a uma matriz de pesos criada pela camada) que serão convoluídos no volume. Durante os testes se mostrou mais eficaz aumentar de 32 para 64 a quantidade de filtros durante as últimas camadas.

Uma camada adicional de otimização é adicionada, a função *DropOut* redefine aleatoriamente algumas unidades de entrada (valor_unidade = 0), o que ajuda a evitar que o modelo treinado se restrinja demais aos dados (*overfitting*) e não consigam se adaptar a situações diferentes.

Figura 9: Camada totalmente conectada

Convertendo as matrizes em arrays

```
In [13]: model.add(Flatten())
```

Camada totalmente conectada

```
In [14]: model.add(Dense(units=512, activation='relu'))
```

```
In [15]: model.add(Dense(units=7, activation='softmax'))
```

Compilando o modelo

```
In [16]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Fonte: Autor

Já na figura 9, vemos que o modelo passa então por um processo de achatamento (*flatten*) onde matrizes com várias linhas são transformadas em arrays de uma única linha, preparando-as para as camadas totalmente conectadas. A primeira camada conectada é uma constituída de 512 neurônios conectados e com ativação ReLU, e em seguida por uma camada de saída com 7 neurônios (o mesmo números de letras que serão reconhecidas) e ativação softmax que tem a função de transformar os valores em números entre 0 e 1 para facilitar a definição do resultado. Depois disso o modelo é compilado com parâmetros para exibir as métricas de acurácia.

7.2.2 Preparação dos dados para treino

Após a definição da rede neural convolucional ser concluída, é necessário preparar os dados para alimentá-la. Conforme vemos na imagem 10, isso é feito com algumas funções da biblioteca Keras como *ImageDataGenerator* que irá ler o arquivo de mídia digital e transformá-lo em um lote de dados para ser processado e *flow_from_directory* que percorre os diretórios atrás de imagens e usa a hierarquia de arquivos como definição das categorias dos objetos.

Figura 10: Preparação dos dados

Preparando os dados para treino

Constroi um conjunto de dados com arquivos de imagem (treino)

```
In [ ]: train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=25,
                                         shear_range=0.5,
                                         zoom_range=0.5,
                                         width_shift_range=0.2,
                                         height_shift_range=0.2,
                                         horizontal_flip=True,
                                         fill_mode="nearest",
                                         validation_split=0.1)
```

Constroi um conjunto de dados com arquivos de imagem (teste)

```
In [ ]: test_datagen = ImageDataGenerator(rescale=1./255)
```

Definindo localização e formato das imagens para treino

```
In [ ]: training_set = train_datagen.flow_from_directory('dataset/treino',
                                                       target_size=(64, 64),
                                                       color_mode='grayscale',
                                                       class_mode='categorical')
```

Definindo localização e formato das imagens para teste

```
In [ ]: test_set = test_datagen.flow_from_directory('dataset/teste',
                                                   target_size=(64, 64),
                                                   color_mode='grayscale',
                                                   class_mode='categorical')
```

Fonte: Autor

Durante a geração dos conjunto de dados (*dataset*), algumas funções são acrescentadas para gerar distorções na imagem durante o treinamento, por exemplo, a imagem é rotacionada(*rotation*), invertida(*horizontal_flip*) ou esticada (*width_shift*), essa técnica chama *data augmentation* melhora a identificação dos objetos após o treinamento.

7.2.3 Parâmetros de treinamento

Com os dados definidos, aprimorados e carregados, chegamos à última parte do algoritmo onde inserimos os últimos parâmetros de aprendizagem. Na função *fit()*, mostrada na figura 11, o primeiro parâmetro diz respeito à localização do *dataset* de treino, definido no passo anterior, na mesma figura 11, o parâmetro *epochs*

determina a quantidade de vezes que o conjunto de dados inteiro será percorrido pelo treinamento. O parâmetro *shuffle* mistura a ordem dos dados de entrada e *validation_data* especifica a localização do *dataset* de teste.

Figura 11: Parâmetros de treinamento

Definindo os parametros de aprendizagem

```
In [ ]: model.fit(
    training_set,
    epochs=50,
    shuffle=True,
    validation_data=test_set)
```

Salvando o modelo

```
In [ ]: model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

Salvando os pesos

```
In [ ]: model.save_weights('model.h5')
```

Fonte: Autor

Por fim, tendo o treinamento sido concluído com sucesso, serão gerados dois arquivos, o arquivo *model.json* com as configurações do modelo e o arquivo *model.h5* com os pesos (*weights*) definidos durante o treinamento, e que irão juntos ser usados durante a fase de reconhecimento (*predict*).

7.3 Algoritmo de reconhecimento

7.3.1 Carregando os modelos gerados

Iniciamos o algoritmo de reconhecimento, correspondente à fase de previsão (*predict*) com a importação das bibliotecas que serão usadas e o carregamento do modelo criado na fase anterior, como visto na figura 12 abaixo.

Figura 12: Carregando os modelos gerados

Importando as bibliotecas necessárias

```
In [ ]: from keras.models import model_from_json
import operator
import cv2
```

Carregando os dados treinados

Carregando o modelo

```
In [ ]: json_file = open("model.json", "r")
model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(model_json)
```

Carregando os pesos no modelo

```
In [ ]: loaded_model.load_weights("model.h5")
```

Fonte: Autor

7.3.2 Captura de vídeo para o reconhecimento

Em seguida, criamos uma variável chamada ‘cap’ que será a instância da função de captura de vídeo da biblioteca *OpenCV*. Também criamos um dicionário que tem como chave os números 0-6 e com os valores das categorias que escolhemos durante o treinamento, que são as letras U-N-I-V-E-S-P. Uma observação é que durante a função *flow_from_directory* do passo anterior, as categorias criadas a partir da hierarquia de arquivos é feita em ordem alfabética, por isso o dicionário também é criado nessa ordem para corresponder com a numeração usada no treino.

O próximo passo envolve um laço de repetição do tipo *while* e por conta disso o algoritmo tem de estar indentado dentro dele para funcionar corretamente e não pode ser separado em blocos como fizemos anteriormente. Pon conta disso, ele está reunido em um único bloco, mostrado na figura 13.

Figura 13: Bloco de código em laço de repetição

Enquanto houver captura de vídeo, faça:

Por conta do laço de repetição (while) não foi possível separar o código em blocos.

```
In [ ]: while True:
    _, frame = cap.read()

    # Transladando a imagem horizontalmente (espelho)
    frame = cv2.flip(frame, 1)

    # Declarando variáveis de coordenada da zona de interesse
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])

    # Desenhando a zona de interesse (retângulo verde)
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (0,255,0) ,1)

    # Instanciando zona de interesse
    captura = frame[y1:y2, x1:x2]

    # Manipulando a imagem dentro da zona de interesse
    captura = cv2.resize(captura,(64, 64))
    captura = cv2.cvtColor(captura, cv2.COLOR_BGR2GRAY)

    # Exibindo a imagem capturada
    cv2.imshow("Captura", captura)

    # Reconhecimento com Inteligencia Artificial

    # Previsão do resultado com base no modelo neural
    result = loaded_model.predict(captura.reshape(1, 64, 64, 1))
    prediction = {'E': result[0][0],
                  'I': result[0][1],
                  'N': result[0][2],
                  'P': result[0][3],
                  'S': result[0][4],
                  'U': result[0][5],
                  'V': result[0][6]}

    # Identificando qual a previsão mais próxima
    prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

    # Exibindo a previsão
    cv2.putText(frame, prediction[0][0], (100, 120), cv2.FONT_HERSHEY_PLAIN, 3, (14,14,14) , 3)
    cv2.imshow("Frame", frame)

    # Interromper com a tecla ESC
    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27:
        break

    # Finaliza captura de vídeo e fecha as janelas
    cap.release()
    cv2.destroyAllWindows()
```

Fonte: Autor

No entanto, é possível acompanhar o processo com os comentários inseridos no algoritmo durante o desenvolvimento. Conforme vemos na figura 14 abaixo, no início do bloco de código temos a inicialização da janela de captura, chamamos ela de *frame*, um pouco abaixo invertemos a imagem com a função *flip* logo que a captura da imagem é invertido, efeito espelho.

Figura 14: Inicialização da captura de vídeo

```
In [ ]: while True:
    _, frame = cap.read()

    # Transladando a imagem horizontalmente (espelho)
    frame = cv2.flip(frame, 1)
```

Fonte: Autor

7.3.3 Definindo uma zona de interesse

Em seguida na figura 15, observamos que o algoritmo desenha um quadrado verde que servirá de referência para a zona de interesse, de onde a imagem para reconhecimento será capturada, por conta disso ele está nomeado como captura.

Figura 15: Criando zona de interesse

```
# Declarando variáveis de coordenada da zona de interesse
x1 = int(0.5*frame.shape[1])
y1 = 10
x2 = frame.shape[1]-10
y2 = int(0.5*frame.shape[1])

# Desenhando a zona de interesse (retângulo verde)
cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (0,255,0) ,2)

# Instanciando zona de interesse
captura = frame[y1:y2, x1:x2]

# Manipulando a imagem dentro da zona de interesse
captura = cv2.resize(captura,(64, 64))
captura = cv2.cvtColor(captura, cv2.COLOR_BGR2GRAY)

# Exibindo a imagem capturada
cv2.imshow("Captura", captura)
```

Fonte: Autor

A retângulo tem por tamanho 64 pixels de largura e altura e seu modo de cor é escala de cinza.

7.3.4 Reconhecimento com Inteligência Artificial

Na figura 16 abaixo, vemos o uso do modelo carregado atribuído a uma variável chamada *result*, dentro é inserida a função *predict* da biblioteca *Keras*, e é

ela que irá fazer a previsão do resultado mais provável entre o modelo treinado e a imagem captada em tempo real. Essa variável é chamada dentro de um dicionário chamado *prediction*, que contém também as letras U-N-I-V-E-S-P em ordem alfabética conforme o número associado à sua categoria durante o treino.

Figura 16: Função de previsão

```
# Reconhecimento com Inteligencia Artificial

# Previsão do resultado com base no modelo neural
result = loaded_model.predict(captura.reshape(1, 64, 64, 1))
prediction = {'E': result[0][0],
              'I': result[0][1],
              'N': result[0][2],
              'P': result[0][3],
              'S': result[0][4],
              'U': result[0][5],
              'V': result[0][6]}

# Identificando qual a previsão mais próxima
prediction = sorted(prediction.items(),
                     key=operator.itemgetter(1),
                     reverse=True)
```

Fonte: Autor

Um pouco abaixo na mesma figura 16, note a função *sorted* que pertence à própria linguagem *Python*, ela irá classificar os itens do dicionário em ordem decrescente (denotado no parâmetro *reverse*) para encontrar qual das categorias tem o maior valor de probabilidade de estar certa.

7.3.5 Exibição do resultado

No último trecho de código apresentado na figura 17, temos a exibição da previsão armazenada na variável *prediction* renderizada em tempo real para o usuário por meio da função *putText()* da biblioteca *OpenCV*.

A variável *interrupt* contém o comando *awaitKey()* que verifica o pressionamento da tecla e a estrutura condicional *if* verifica se o valor da tecla pressionada é igual ao valor associado à tecla ESC no teclado (0xFF está em formato Hexadecimal). Se pressionada é executado um *break* que sai do laço de repetição *while* e interrompe a execução dos algoritmos deste bloco.

Figura 17: Exibição do resultado

```

# Exibindo a previsão
cv2.putText(frame, prediction[0][0], (100, 120),
            cv2.FONT_HERSHEY_PLAIN,
            3, (14,14,14), 3)
cv2.imshow("Frame", frame)

# Interromper com a tecla ESC
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == 27:
    break

# Finaliza captura de vídeo e fecha as janelas
cap.release()
cv2.destroyAllWindows()

```

Fonte: Autor

Por fim, ao fim do código mostrado na figura 17, a função *release()* libera o recurso de vídeo que estava em uso e a função *destroyAllWindows()* irá finalizar as janelas abertas da câmera e da zona de interesse (*frame* e *captura* respectivamente).

8. RESULTADOS

8.1 Algoritmo de renderização de gráfico

Para avaliar os resultados do algoritmo foram realizados vários testes com parametrizações diferentes, os testes foram realizados no algoritmo de treinamento. Também foi desenvolvido um algoritmo para o desenho de gráficos com intuito de auxiliar na visualização e comparação do desempenho de cada fase. Na figura 18, vemos o algoritmo responsável por plotar, ou desenhar, esses gráficos.

Figura 18: Função para criação de gráficos

```
In [ ]: import matplotlib.pyplot as plt
```

Renderização de gráfico de acurácia

```
In [ ]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Precisão do Modelo')
plt.ylabel('Precisão')
plt.xlabel('Época')
plt.legend(['Treino', 'Teste'], loc='upper left')
plt.show()
```

Renderização de gráfico de perda

```
In [ ]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Perda do Modelo')
plt.ylabel('Perda')
plt.xlabel('Épocas')
plt.legend(['Treino', 'Teste'], loc='upper left')
plt.show()
```

Fonte: Autor

8.2 Modificações no dataset (*Data Augmentation*)

As modificações pré-processamento, também chamadas de *Data Augmentation*, cria automaticamente imagens de treinamento com diferentes modificações, como rotação aleatória, inversão, *zoom*, alteração em saturação, contraste, etc. Essa técnica visa aumentar a quantidade de amostras para melhorar a qualidade do treinamento. Foi configurado um modelo com as seguintes modificações ativas: cisalhamento, zoom, deslocamento de largura, deslocamento de altura, inversão horizontal, modo de preenchimento, divisão de validação, conforme mostrado na figura 19, e realizado um treinamento de 15 épocas (*epochs*) com e sem essas modificações, a fim de comparar os resultados.

Figura 19: Dataset com *Image Augmentation*

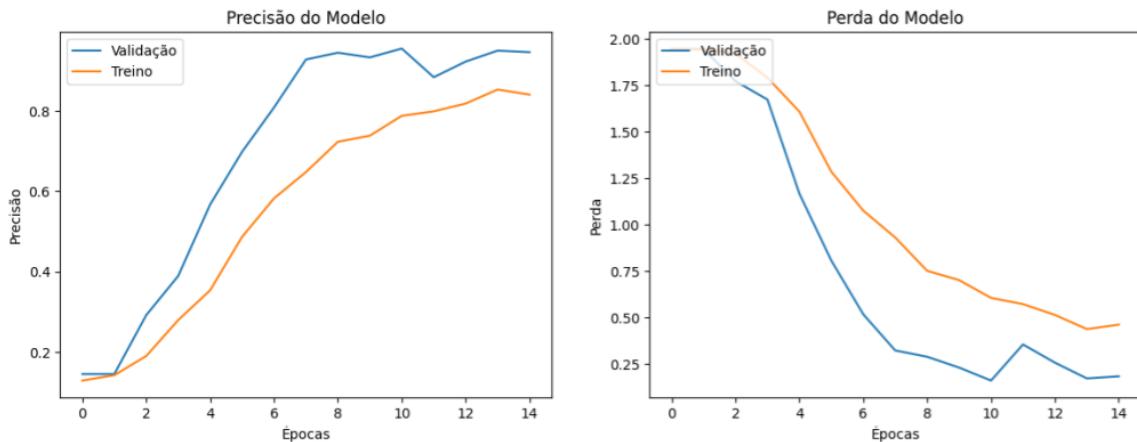
Constroi um conjunto de dados com arquivos de imagem (treino)

```
In [ ]: train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=25,
                                         shear_range=0.5,
                                         zoom_range=0.5,
                                         width_shift_range=0.2,
                                         height_shift_range=0.2,
                                         horizontal_flip=True,
                                         fill_mode="nearest",
                                         validation_split=0.1)
```

Fonte: Autor

O resultado obtido do treinamento com as modificações foi de 94,55% de precisão no teste de validação e altos 17,21% de perda ao fim do treinamento, abaixo é possível ver no gráfico 1, o andamento do treinamento:

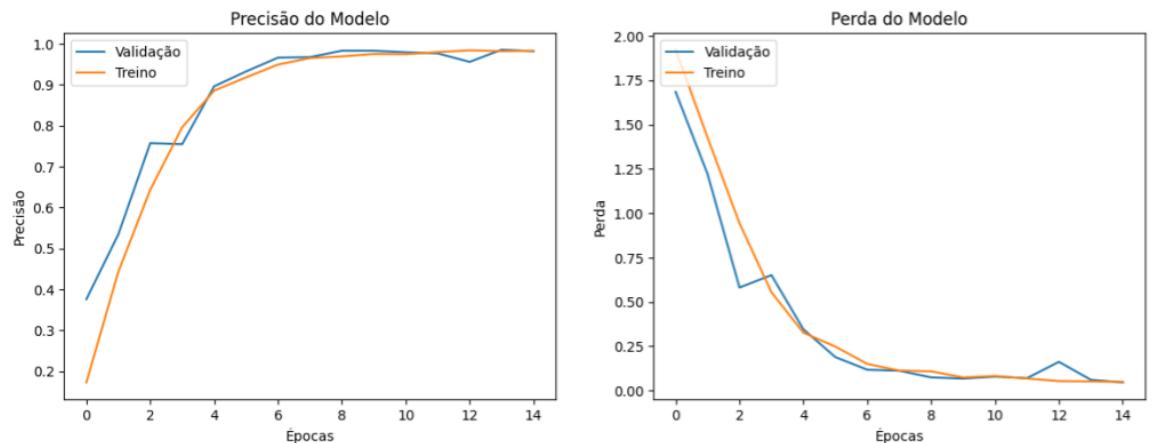
Gráfico 1 - Resultados com DataAugmentation



Fonte: Autor

Um segundo treinamento sem as modificações foi feito e o resultado foi 98,57% de precisão no teste de validação e apenas 4,60% de perda atingido na última época (*epoch*), abaixo é possível ver no gráfico 2 o andamento do treinamento:

Gráfico 2 - Resultados sem Data Augmentation



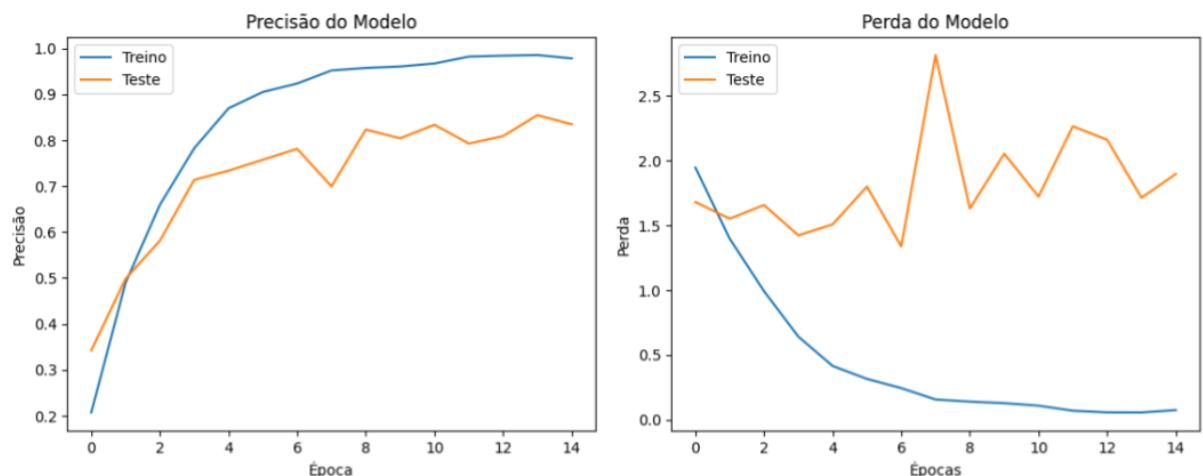
Fonte: Autor

Os resultados mostrados nos gráficos e a diferença entre a porcentagem de perda, nos levam a concluir que neste caso a técnica de *data augmentation* teve um resultado negativo no modelo, sendo assim, a técnica não foi usada no projeto.

8.3 Limpeza do dataset

Após o ajuste acima no algoritmo, efetuou-se testes com tamanho de 15 épocas (*epochs*) e quantidades diferentes de camadas de convolução, todas com a mesma parametrização. O treinamento com 2 camadas, apresentado no gráfico 3, obteve no teste a precisão máxima de 85,43%, no entanto à direita vemos um pico de perda máxima de incríveis 281,59%. Concluiu-se com este resultado que algo estava errado.

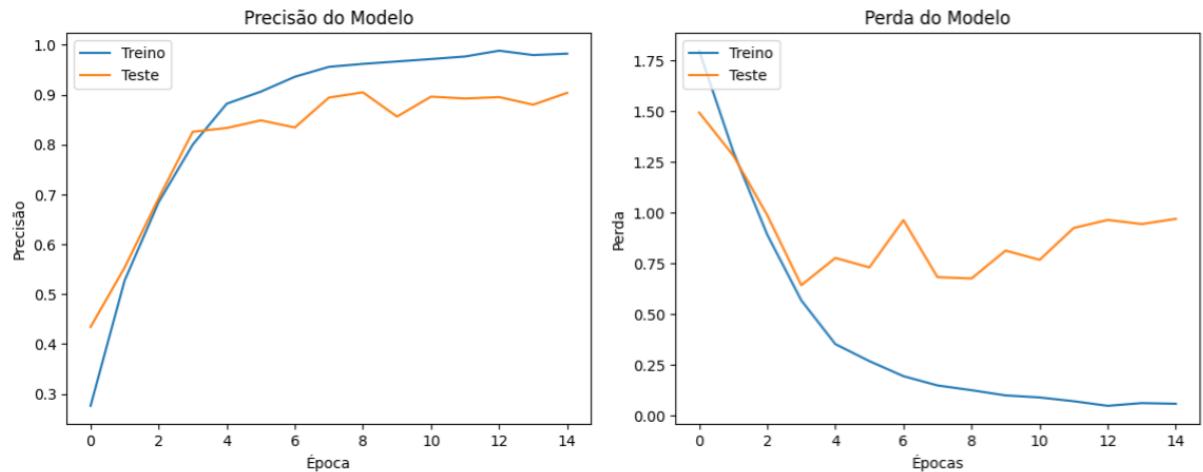
Gráfico 3 - Treinamento com 2 camadas e dataset sujo



Fonte: Autor.

O treinamento com 3 camadas apresentou durante o teste, a precisão máxima de 90,3%, porém a perda se manteve na faixa de 90% infelizmente. O gráfico 4 a seguir ilustra essa situação:

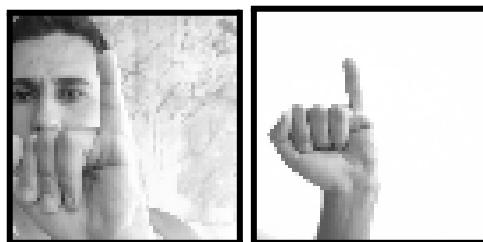
Gráfico 4 - Treinamento com 3 camadas e *dataset* sujo



Fonte: Autor.

Como mostram os gráficos, o algoritmo estava com dificuldade de melhorar o desempenho durante os testes de validação. Foi feita uma avaliação no *dataset* de imagens de teste e foi constatado que haviam algumas imagens que continham mais do que apenas as sinalizações necessárias, conforme ilustra a figura 20 a seguir, à esquerda uma imagem suja que atrapalha o algoritmo e à direita uma imagem limpa.

Figura 20: Comparação de amostra suja e limpa

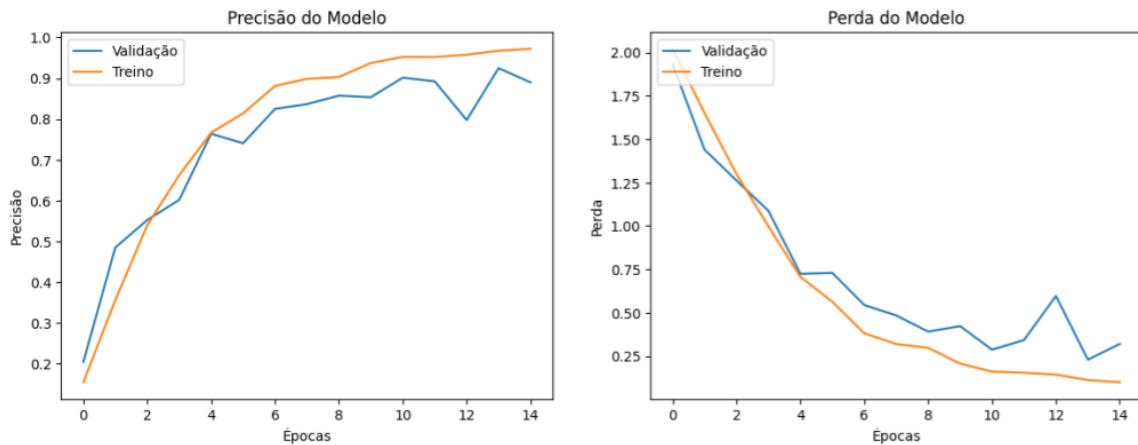


Fonte: Autor.

Foi feita a limpeza dos dados no *dataset* de imagens de teste e refeito os treinamentos com 2 e 3 camadas. Durante o teste de validação, o treinamento com 2 camadas demonstrado no gráfico 5, chegou à precisão máxima de 92,48% e a perda na penúltima época foi de apenas 22,89%. A perda durante o teste de validação teve

uma diminuição de aproximadamente 92%, contrastado pelos treinamentos anteriores com o *dataset* sujo, um resultado surpreendente.

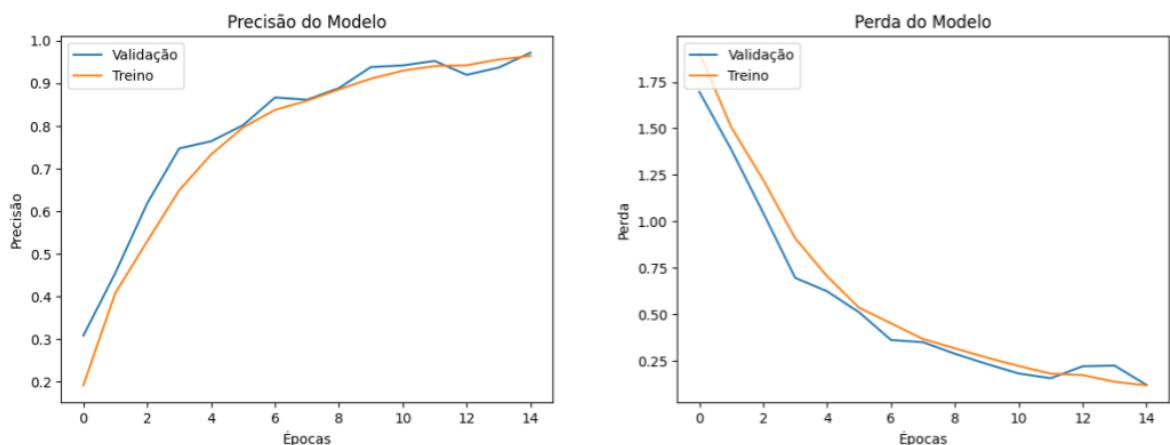
Gráfico 5 - Treinamento com 2 camadas e *dataset* limpo



Fonte: Autor.

Enfim, no gráfico 6 vemos que o treinamento com 3 camadas alcançou durante o teste de validação a precisão máxima de 97,15% e perda na última época reduzida a 11,91%. Uma redução de 87,70% na perda de dados durante a validação, um resultado muito bom também. O tempo de treino em ambos os testes foi de 100 segundos.

Gráfico 6 - Treinamento com 3 camadas e *dataset* limpo



Fonte: Autor.

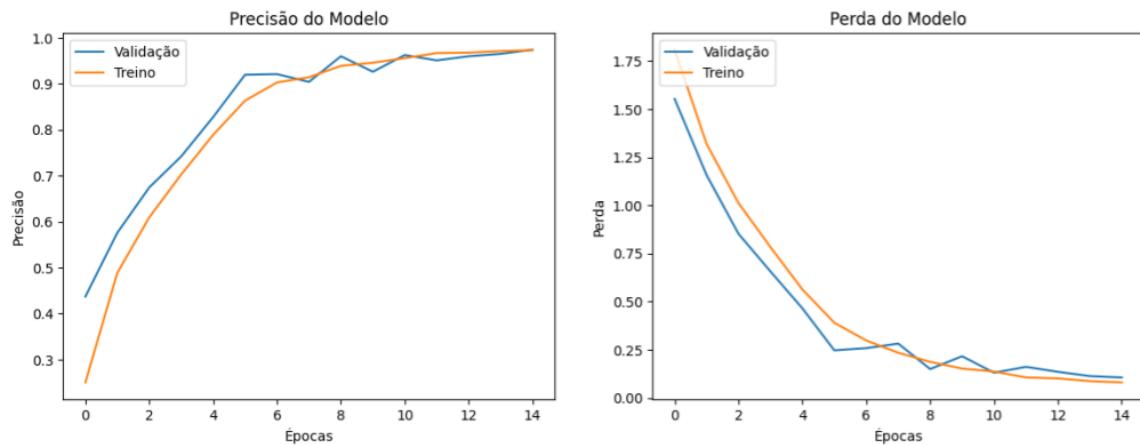
Essas experiências nos ajudaram a provar que um conjunto de dados limpo é determinante para o sucesso do algoritmo de aprendizado de máquina.

8.4 Camadas de convolução

Durante os treinamentos anteriores foi possível notar uma diferença entre os resultados dos modelos de 2 e 3 camadas, para compreender como alcançar o melhor desempenho foi feito um teste com 4 camadas de convolução para ser comparada com os dados dos dois últimos treinos.

O treinamento com 4 camadas, apresentado abaixo no gráfico 7, atingiu durante o teste precisão máxima 97,41% e a perda na última época foi reduzida a 10,59%. Uma pequena melhora em relação ao teste com 3 camadas, o tempo de treino foi o mesmo.

Gráfico 7- Treinamento com 4 camadas



Fonte: Autor.

Um teste com 5 camadas não seria tão necessário em vista do ganho tão pequeno de desempenho que já havia sido registrado entre 3 e 4 camadas de convolução, no entanto, para confirmação foram feitas diversas tentativas, mas todas resultaram em erros da biblioteca Keras, a depuração e estudo da causa dos erros demandaria um tempo não disponível no momento em vista do custo-benefício para o projeto.

Abaixo na figura 21, uma captura de tela com um dos erros ao tentar compilar e rodar a rede neural com 5 camadas:

Figura 21: Erros aos adicionar quinta camada

Quinta camada de convolução e pooling

```
In [12]: model.add(Convolution2D(64, (3, 3), activation='relu'))

-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20728\1545351636.py in <cell line: 1>()
----> 1 model.add(Convolution2D(64, (3, 3), activation='relu'))

~\Documents\GitHub\reconhecedor-libras\venv\lib\site-packages\tensorflow\python\trackable\base.py in _method_wrapper(self, *args, **kwargs)
    203     self._self_setattr_tracking = False # pylint: disable=protected-access
    204     try:
--> 205         result = method(self, *args, **kwargs)
    206     finally:
    207         self._self_setattr_tracking = previous_value # pylint: disable=protected-access

~\Documents\GitHub\reconhecedor-libras\venv\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
    68     # To get the full stack trace, call:
    69     # `tf.debugging.disable_traceback_filtering()`
--> 70     raise e.with_traceback(filtered_tb) from None
    71 finally:
    72     del filtered_tb

~\Documents\GitHub\reconhecedor-libras\venv\lib\site-packages\tensorflow\python\framework\ops.py in _create_c_op(graph, node_def, inputs, control_inputs, op_def, extract_traceback)
    1967     except errors.InvalidArgumentError as e:
    1968         # Convert to ValueError for backwards compatibility.
-> 1969         raise ValueError(e.message)
    1970     # Record the current Python stack trace as the creating stacktrace of this

ValueError: Exception encountered when calling layer "conv2d_4" (type Conv2D).

Negative dimension size caused by subtracting 3 from 2 for '{{node conv2d_4/Conv2D}} = Conv2D[T=DT_FLOAT, data_format="NHWC", dilations=[1, 1, 1, 1], explicit_paddings=[], padding="VALID", strides=[1, 1, 1, 1], use_cudnn_on_gpu=true](Placeholder, conv2d_4/Conv2D/ReadVariableOp)' with input shapes: [?,2,2,64], [3,3,64,64].
```

Call arguments received by layer "conv2d_4" (type Conv2D):
• inputs=tf.Tensor(shape=(None, 2, 2, 64), dtype=float32)

Fonte: Autor.

Desta forma, se justifica a decisão de definir para este projeto a configuração de 4 camadas convolucionais sem *data augmentation*, juntamente com as respectivas camadas de *pooling*, normalização e funções de ativação. Esses foram apenas os resultados dos testes mais significativos para este documento, no entanto, foram realizados dezenas de testes e empenhadas dezenas de horas de trabalho computacional durante os treinamentos da rede neural artificial que exigem alto custo computacional.

8.5 Reconhecimento em tempo real

As imagens abaixo, na figura 22, mostram o funcionamento do modelo de aprendizado pronto, conforme analisado em pormenores na seção 7.3 deste projeto, ele irá iniciar uma captura de vídeo em tempo real e comparar com o modelo treinado carregado.

Figura 22: Reconhecimento em tempo real



Fonte: Autor.

O algoritmo reconhece apenas o que está dentro da zona de interesse (quadrado verde), e desconsidera os dados advindos do restante da captação de imagem que vemos na figura 22. No canto superior esquerdo é renderizado o resultado do processo de previsão (*predict*) da rede neural convolucional.

9. CONSIDERAÇÕES FINAIS

Observamos com os resultados um nível de precisão muito satisfatório, acima dos 90% de acurácia, essa é uma evidência do quanto bem sucedidos são os conceitos de inteligência artificial empregados atualmente, tanto no campo conceitual e teórico com as suas equações e diversas abordagens possíveis de acordo com a necessidade, quanto também no campo prático com bibliotecas completas de fácil implementação.

Os avanços na área de arquitetura computacional tem sido decisivo para o sucesso atual do campo de inteligência artificial, uma vez que as técnicas de aprendizagem demandam alto poder e custo computacional. Um exemplo são os avanços no desenvolvimento de novas tecnologias de *GPUs (Graphic Processor Unit)* mais eficientes e potentes impulsionados pelo setor de jogos e cinematografia, e a criação de dispositivos de circuito integrado de aplicação específica, conhecidos como *ASIC (Application Specific Integrated Circuit)* como o *TPU (Tensor Processor Unit)* desenvolvido pelo Google que são projetados para o uso específico e único de aprendizado de máquina. (WANG et al., 2019)

Este pequeno projeto comprova o funcionamento e eficiência da rede neural convolucional. No entanto, há agora uma ampla janela de oportunidades de estudos visando o aperfeiçoamento do algoritmo para a elevação de seu desempenho e acuracidade. Nesse quesito, a abertura do código das bibliotecas usadas no campo da inteligência artificial tem contribuído para seu crescimento rápido, logo que mais pessoas têm a oportunidade de contribuir no desenvolvimento e estudo de novos algoritmos, conceitos e equações.

Uma continuação deste projeto pode ser feito com um rico *dataset* apresentado em um trabalho brasileiro de 2019 chamado “Desenvolvimento de uma Base de Dados de Sinais de Libras para Aprendizado de Máquina” (CASTRO et al., 2019) apresentado no 14º Simpósio Brasileiro de Automação Inteligente sediado na cidade de Ouro Preto - Minas Gerais. (SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO, 2019).

A língua de sinais brasileira é muito rica em gestos e expressões, outra abordagem seria o uso de uma técnica de aprendizado de máquina que consiga captar a imagem em três dimensões seria uma solução para uma tecnologia capaz de compreender não somente os gestos mas também as expressões faciais e a profundidade dos gestos, se distantes ou próximos do visualizador.

REFERÊNCIAS

BARCA, Maria C. S. et al. **Treinamento de Redes Neurais Artificiais: O Algoritmo Backpropagation.** In: IX Encontro Latino Americano de Iniciação Científica e V Encontro Latino Americano de Pós-graduação, Universidade do Vale do Paraíba. Vale do Paraíba: Univap; 2005. p. 46-49. Disponível em: http://www.inicepg.univap.br/cd/INIC_2005/inic/IC1%20anais/IC1-17.pdf. Acesso em: 16 nov. 2022.

BRASIL. Lei nº 10.436, de 24 de abril de 2002. Dispõe sobre a Língua Brasileira de Sinais - Libras e dá outras providências. **Diário Oficial da União**, Brasília, DF, p. 23, 25 abr. 2002. Disponível em: http://www.planalto.gov.br/ccivil_03/leis/2002/l10436.htm. Acesso em: 8 out. 2022.

BRASIL. Lei nº 12.319, de 1 de setembro de 2010. Regulamenta a profissão de Tradutor e Intérprete da Língua Brasileira de Sinais - LIBRAS. **Diário Oficial da União**: Seção 1, Brasília, DF, ano 169, p. 2, 1 set. 2010. Disponível em: <https://pesquisa.in.gov.br/imprensa/jsp/visualiza/index.jsp?jornal=1&pagina=1&data=02/09/2010>. Acesso em: 8 out. 2022.

BRINGSJORD, Selmer et al. Artificial Intelligence. **The Stanford Encyclopedia of Philosophy**, [s. l.], 12 Jul. 2018. Disponível em: <https://plato.stanford.edu/archives/fall2022/entries/artificial-intelligence/#WhatExacAI>. Acesso em: 3 nov. 2022.

CARVALHO, André P. L. **Redes Neurais Artificiais: Perceptron Multi-Camadas (MLP)**. [s. l.], 2009. Disponível em: <https://sites.icmc.usp.br/andre/research/neural/MLP.htm>. Acesso em: 16 nov. 2022.

CASTRO, GIULIA ZANON et al. Anais do 14º Simpósio Brasileiro de Automação Inteligente, 14., 2019, Ouro Preto - MG. **Desenvolvimento de uma Base de Dados de Sinais de Libras para Aprendizado de Máquina: Estudo de Caso com CNN 3D [...]**. Ouro Preto - MG: [s. n.], 2019. v. 1. DOI 10.17648/sbai-2019-111451. Disponível em: <https://proceedings.science/sbai-2019/papers/desenvolvimento-de-uma-base-de-dados-de-sinais-de-libras-para-aprendizado-de-maquina--estudo-de-caso-com-cnn-3d>. Acesso em: 7 out. 2022.

DARTMOUTH WORKSHOP. In **Wikipedia, The Free Encyclopedia**. Disponível em: https://en.wikipedia.org/w/index.php?title=Dartmouth_workshop&oldid=1115555838. Acesso em: 4 nov. 2022.

DEEPMIND. **AlphaGo: The matches**. [s. l.], 2022. Disponível em: <https://www.deepmind.com/research/highlighted-research/alphago>. Acesso em: 4 nov. 2022.

DELOHERY, Caitlin. These three companies are making self-driving cars safer. **Utah Business**, [s. l.], p. 1, 25 out. 2022. Disponível em:

<https://www.utahbusiness.com/are-self-driving-cars-safe-companies-in-car-safety/>. Acesso em: 3 nov. 2022.

EDA, Kavlkoglu. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?. **Artificial Intelligence IBM BLog**, [S. I.], 27 maio 2020. Disponível em: <https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>. Acesso em: 16 nov. 2022.

GÉRON , Aurélien. **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow**. 2ª Edição. [S. I.]: Alta Books, 2021. cap. 1, p. 3-27. ISBN 978-85-5081-548-0.

GOOGLE. Google Translate. In: **Translate**: Explore the world in over 100 languages.. [S. I.], 2022. Disponível em: <https://translate.google.com/intl/en/about/languages/>. Acesso em: 4 nov. 2022.

GOTTFREDSON, Linda S. Intelligence. **Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography**, [s. I.], v. 24, n. 1, p. 13-23, 1997. DOI [https://doi.org/10.1016/S0160-2896\(97\)90011-8](https://doi.org/10.1016/S0160-2896(97)90011-8). Disponível em: <https://www1.udel.edu/educ/gottfredson/reprints/1997mainstream.pdf>. Acesso em: 3 nov. 2022.

HOPE, Graham. Volkswagen Invests \$2.3B to Deploy Autonomous Vehicles in China. **IoT World Today**, [S. I.], p. 1, 21 out. 2022. Disponível em: <https://www.iotworldtoday.com/2022/10/21/volkswagen-invests-2-3b-to-deploy-autonomous-vehicles-in-china/>. Acesso em: 3 nov. 2022.

IBGE, INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Estatísticas de Gênero: Total de pessoas com deficiência auditiva no Brasil**. [S. I.], 2010. Disponível em: <https://www.ibge.gov.br/apps/snig/v1/index.html?loc=0>. Acesso em: 7 out. 2022.

KAY, Steven M. Fundamentals of Statistical Signal Processing: Detection theory. In: STATISTICAL Decision Theory I. [S. I.]: Prentice-Hall PTR, 1998. v. 2, ISBN ISBN-13: 9780135041352.

LECUN, Yann et al. Gradient-Based Learn Applied to Document Recognition. **Proceedings of the IEEE**, [s. I.], Nov. 1998. Disponível em: <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>. Acesso em: 16 nov. 2022.

MCCULLOCH, WARREN S.; PITTS, WALTER. A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. **BULLETIN OF MATHEMATICAL BIOPHYSICS** , [s. I.], v. 5, p. 115-133, 1943. Disponível em: <https://home.csulb.edu/~cwallis/382/readings/482/mcculloch.logical.calculus.ideas.1943.pdf>. Acesso em: 4 nov. 2022.

MITCHELL, Tom M. **Machine Learning**. [S. I.: s. n.], 1997. 432 p. ISBN 0-07-042807-7.

NASCIMENTO, E. O.; OLIVEIRA, L. N. Sensitivity Analysis of Cutting Force on Milling Process using Factorial Experimental Planning and Neural Networks. **IEEE Latin America Transactions**, vol. 14, n. 12, p. 4811-4820, 2016. Disponível em: https://www.researchgate.net/publication/312027170_Sensitivity_Analysis_of_Cutting_Force_on_Milling_Process_using_Factorial_Experimental_Planning_and_Artificial_Neural_Networks. Acesso em: 16 nov. 2022.

NEISSE , Ulric et al. Intelligence: Knowns and Unknowns. **American Psychologist**, [s. l.], n. 51, p. 77–101, 1996. DOI <https://psycnet.apa.org/doi/10.1037/0003-066X.51.2.77>. Disponível em: http://psych.colorado.edu/~carey/pdfFiles/IQ_Neisser2.pdf. Acesso em: 3 nov. 2022.

PAULY, Leo et al. Deeper Networks for Pavement Crack Detection. **34th International Symposium in Automation and Robotics in Construction**, Taipei, Taiwan, 2017. DOI 10.22260/ISARC2017/0066. Disponível em: https://www.researchgate.net/publication/319235847_Deeper_Networks_for_Pavement_Crack_Detection. Acesso em: 16 nov. 2022.

SARACCO, Roberto. Congrats Xiaoyi. You are now a medical doctor. **IEEE Future Directions**, [S. l.], 2 dez. 2017. Blog, p. 1. Disponível em: <https://cmte.ieee.org/futuredirections/2017/12/02/congrats-xiaoyi-you-are-now-a-medical-doctor/>. Acesso em: 4 nov. 2022.

SETHUNYA, Joseph et al. Natural Language Processing: A Review. **International Journal of Research in Engineering and Applied Sciences**, [s. l.], v. 6, ed. 3, p. 207-212, Março 2016. Disponível em: https://www.researchgate.net/profile/Sethunya-Joseph/publication/309210149_Natural_Language_Processing_A_Review/links/5805ea1f08ae03256b75d965/Natural-Language-Processing-A-Review.pdf. Acesso em: 4 nov. 2022.

SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO. 14º Simpósio Brasileiro de Automação Inteligente. In: **14º Simpósio Brasileiro de Automação Inteligente - SBAI 2019**. Ouro Preto - MG, Outubro 2019. Disponível em: <https://www.sba.org.br/Proceedings/SBAI/SBAI2019/evento.html>. Acesso em: 8 out. 2022.

ROSENBLAT, F. **The perceptron, a perceiving and recognizing automaton - Project Para**. Report n.º 85-460-1. January 1957. Cornell Aeronautical Laboratory.

TURING, Alan. COMPUTING MACHINERY AND INTELLIGENCE. **Mind**, [s. l.], v. LIX, n. 236, p. 433-460, 10 1950. DOI 10.1093/mind/LIX.236.433. Disponível em: <https://academic.oup.com/mind/article/LIX/236/433/986238>. Acesso em: 3 nov. 2022.

UFRGS - UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. Instituto de Matemática e Estatística. **As propriedades de translação e da transformada da integral: A função de Heaviside**. [S. l.], 26 jul. 2022. Disponível em: [https://www.ufrgs.br/reamat/TransformadasIntegrals/livro-tl/aptdedtd-a_funx00e7x00e3o_de_heaviside.html](https://www.ufrgs.br/reamat/TransformadasIntegrais/livro-tl/aptdedtd-a_funx00e7x00e3o_de_heaviside.html). Acesso em: 16 nov. 2022.

VIJAY, Laxman. **Linear Algebra explained in the context of deep learning.** [S. I.], 4 dez. 2018. Disponível em: <https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494>. Acesso em: 16 nov. 2022.

WANG, Yu Emma et al. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. **John A. Paulson School of Engineering and Applied Sciences Harvard University**, [s. I.], p. 1-10, Julho 2019. Disponível em: <https://ui.adsabs.harvard.edu/abs/2019arXiv190710701W>. Acesso em: 26 nov. 2022.

WHO, WORLD HEALTH ORGANIZATION. Deafness and hearing loss. **Fact sheets**, [s. I.], 1 abr. 2021a. Disponível em: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>. Acesso em: 7 out. 2022.

WHO, WORLD HEALTH ORGANIZATION. Deafness and hearing loss. **Health topics**, [s. I.], Abril 2021b. Disponível em: https://www.who.int/health-topics/hearing-loss#tab=tab_1. Acesso em: 7 out. 2022.

WHO, WORLD HEALTH ORGANIZATION. WHO: 1 in 4 people projected to have hearing problems by 2050. **News**, [s. I.], 2 mar. 2021c. Disponível em: <https://www.who.int/news/item/02-03-2021-who-1-in-4-people-projected-to-have-hearing-problems-by-2050>. Acesso em: 7 out. 2022.

ZHOU, Y. T.; CHELLAPPA, R. Computation of optical flow using a neural network. **IEEE 1988 International Conference on Neural Networks**, Boston, MA, USA, p. 71-78, 1 set. 1988.