UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

João Lucas Ferreira Silva

Reconhecimento e Geração de Áudio de Sinais de Libras utilizando Visão Computacional e Redes Neurais Convolucionais

Blumenau
2024

João Lucas Ferreira Silva

Reconhecimento e Geração de Áudio de Sinais de Libras utilizando Visão Computacional e Redes Neurais Convolucionais

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Mauri Ferrandin, Dr.

Blumenau
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Ferreira Silva, João Lucas
Visão Computacional e Redes Neurais para tradução de Libras, com Sintetizador de Voz (Text-To-Speech) / João Lucas Ferreira Silva ; orientador, Mauri Ferrandin, 2024.
56 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Visão Computacional. 3. Redes Neurais. 4. Libras. I. Ferrandin, Mauri. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

João Lucas Ferreira Silva

Reconhecimento e Geração de Áudio de Sinais de Libras utilizando Visão Computacional e Redes Neurais Convolucionais

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 27 de Junho de 2024.

Banca Examinadora:

Documento assinado digitalmente
MAURI FERRANDIN
Data: 12/07/2024 08:06:41-0300
CPF: ***.580.869-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mauri Ferrandin Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Carlos Roberto Moratelli
Data: 12/07/2024 08:42:56-0300
CPF: ***.999.369-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Carlos Roberto Moratelli, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Maiquel de Brito
Data: 12/07/2024 09:16:54-0300
CPF: ***.657.150-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Maiquel de Brito, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus queridos pais que me incentivaram e apoiaram durante toda minha trajetória.

RESUMO

A Língua Brasileira de Sinais (LIBRAS) é utilizada por mais de 2 milhões de indivíduos, tornando-se cada vez mais essencial desenvolver tecnologias que auxiliem na inclusão dos surdos e deficientes auditivos na comunicação com o restante da comunidade. O presente trabalho apresenta um modelo computacional que utiliza visão computacional e redes neurais para tradução em tempo real de algumas letras do alfabeto em LIBRAS, integrado com um sintetizador de voz. Foram desenvolvidas e analisadas redes neurais CNN, FFCNN e VGG16, que apresentaram excelentes resultados, com acurácias acima de 99,3%. A rede FFCNN destacou-se com a maior acurácia, alcançando 99,9%. Todo o desenvolvimento foi realizado em plataforma em nuvem, permitindo a execução em máquinas de baixa performance.

Palavras-chave: Libras; Visão Computacional; Aprendizado de Máquina; CNN; VGG16; FFCNN.

ABSTRACT

Brazilian Sign Language (LIBRAS) is used by over 2 million individuals, making it increasingly essential to develop technologies that aid in the inclusion of the deaf and hard of hearing in communication with the rest of the community. This work presents a computational model that utilizes computer vision and neural networks for real-time translation of some letters of the alphabet in LIBRAS, integrated with a voice synthesizer. CNN, FFCNN, and VGG16 neural networks were developed and analyzed, showing excellent results with accuracies above 99.3%. The FFCNN network stood out with the highest accuracy, reaching 99.9%. All development was conducted on a cloud platform, enabling execution on low-performance machines.

Keywords: Libras; Computer Vision; Machine Learnig; CNN; VGG16; FFCNN.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A	Acurácia
ASL	*American Sign Language*
AUC-ROC	Área Sob a Curva
BGR	*Blue-Green-Red* ou Azul-Verde-Vermelho
BSL	*British Sign Language*
CNN	*Convolutional Neural Networks*
ELU	*Exponential Linear Unit*
F1	*F1-Score*
FFCNN	*Feature Fusion-based Convolutional Neural Network*
FN	Falso Negativo
FP	Falso Positivo
IA	Inteligência Artificial
Libras	Língua Brasileira de Sinais
loss	Perda
LSF	*Langue des Signes Française*
LSTM	*Long Short-Term Memory*
MAE	Erro Médio Absoluto
ML	*Machine Learning*
MSE	Erro Quadrático Médio
P	Precisão
R	*Recall*
ReLU	*Rectifier Linear Unit*
RGB	*Red-Green-Blue* ou Vermelho-Verde-Azul
RMSE	Raiz do Erro Quadrático Médio
RN	Rede Neural
RReLU	*Randomized Leaky ReLU*
TN	Verdadeiro Negativo
TP	Verdadeiro Positivo
TTS	*Text-To-Speech*
TTY	*TeleTYpewriter*
VGG	*Visual Geometry Group*
VGG16	*Visual Geometry Group 16 Layers*

SUMÁRIO

1 INTRODUÇÃO

A Libras (Língua Brasileira de Sinais), um sistema gestual-visual utilizado pela comunidade surda no Brasil, substitui a linguagem oral por gestos e expressões faciais. Apresenta uma certa complexidade por ser composta de diferentes níveis linguísticos, expressões e estruturas gramaticais próprias. Segundo dados do IBGE, 5% da população brasileira possui algum problema auditivo, com estimativas de que mais de 2 milhões de indivíduos usam a língua de sinais. Reconhecida pela Lei n.º 10.436/2002 como idioma oficial das Comunidades Surdas Brasileiras, a Libras promove a acessibilidade na comunicação e demanda do poder público medidas de apoio e difusão para garantir sua utilização generalizada.

A inclusão de pessoas surdas na sociedade tem sido um desafio persistente, marcado por obstáculos significativos no acesso à comunicação, educação, saúde e mercado de trabalho. O advento da Lei Brasileira de Inclusão da Pessoa com Deficiência (LBI) em 2015 reflete um reconhecimento crescente da necessidade de políticas que promovam a igualdade de oportunidades para todos os cidadãos, independentemente de suas habilidades auditivas.

É evidente que a comunidade surda enfrenta problemas para acesso a mercado de trabalho, edução, lazer e saúde. Com isso, se faz necessário alternativas de comunicação, como o TTY (*TeleTYpewriter*), que é um equipamento que converte texto em áudio e vice-versa, permitindo que pessoas com deficiência auditiva ou fala possam se comunicar por telefone. Outro exemplo, o Hand Talk, um aplicativo que utiliza realidade virtual para tradução de Libras.

Projetos como a Alexa, que reconhecem gestos de ASL (*American Sign Language*) usando IA e sintetizadores de voz, demonstram o potencial de combinar visão computacional e redes neurais com sintetizadores de voz para a tradução de Libras em tempo real. Outro grande projeto é a IA da Lenovo que captura os movimentos de Libras e gera mensagens de texto e áudio. Ambos projetos mostram que investimentos nesse tema, são rentáveis e necessários.

Entretanto, as pesquisas sobre Libras carecem de desenvolvimentos mais abrangentes devido à sua restrição territorial, ao ser utilizada apenas no Brasil. Essa limitação geográfica impede que muitos avanços tecnológicos e acadêmicos, comuns em outras línguas de sinais mais difundidas globalmente, sejam aplicados diretamente a Libras. Portanto, é crucial fomentar iniciativas que visem a expansão e o aprimoramento do uso da Libras, aproveitando tecnologias emergentes para superar essa barreira territorial e promover uma inclusão mais efetiva da comunidade surda brasileira.

1.1 OBJETIVOS

Este trabalho tem como propósito o estudo e a aplicação de técnicas de visão computacional, redes neurais e TTS (*Text-To-Speech*) para desenvolvimento de um protótipo que auxilia na inclusão da comunidade surda brasileira. A seguir, são apresentados o objetivo geral e os objetivos específicos deste projeto.

1.1.1 Objetivo Geral

Desenvolver um modelo computacional capaz de traduzir sinais da Libras para língua portuguesa, utilizando técnicas de redes neurais para a tarefa de classificação de gestos do alfabeto manual em Libras, com desenvolvimento em plataforma em nuvem, sem a necessidade de hardware local.

1.1.2 Objetivos Específicos

- Estudo bibliográfico de trabalhos relacionados;
- Implementar reconhecimento de padrões de Libras por meio das técnicas de Redes Neurais;
- Utilizar métricas de avaliação, a fim de comparar e analisar desempenho dos algoritmos utilizados;
- Testar o reconhecimento de Libras, integrando a uma webcam para reconhecimento em tempo real;
- Implementar um sintetizador de voz para converter os sinais de Libras reconhecidos em fala.

1.2 ESCOPO

Esta pesquisa contempla o desenvolvimento de um modelo computacional, utilizando técnicas de visão computacional e redes neurais convolucionais, especificamente CNN, FFCNN e VGG16. A pesquisa se limita à tradução de sinais do alfabeto de Libras representados por gestos estáticos, não contemplando gestos dinâmicos. Todo o desenvolvimento foi realizado em uma plataforma em nuvem, eliminando a necessidade de recursos de hardware local, necessitando apenas de conexão com a internet. O *dataset* utilizado é um repositório público disponível na plataforma GitHub, contendo imagens que focam apenas nas mãos, sem incluir o restante do corpo, o que implica que o modelo é treinado exclusivamente para reconhecer gestos manuais isolados. Para o reconhecimento em tempo real, a webcam deve estar posicionada em um ambiente bem iluminado e a uma distância adequada. Idealmente, os sinais devem ser realizados com um fundo monocromático para

facilitar a detecção e reconhecimento das mãos. As principais métricas utilizadas para avaliar a desempenho dos modelos foram acurácia, perda (*loss*), precisão, recall e F1-Score.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho se divide em cinco capítulos. A Capítulo 1 apresenta a motivação e o contexto da pesquisa, seguida pela definição dos objetivos gerais e específicos. O Capítulo 2 aborda a revisão da literatura existente sobre Libras (Língua Brasileira de Sinais), conceitos de aprendizado de máquina, detalhamento de redes neurais, incluindo CNN, FFCNN e VGG16, além de discutir visão computacional, trabalhos correlatos e principais bibliotecas e frameworks utilizados, como *TensorFlow, Keras, Numpy, OpenCV, Web Speech API, Google Colab e MediaPipe*. O Capítulo 3 descreve o *dataset* utilizado, o pré-processamento dos dados, as arquiteturas das redes neurais desenvolvidas, os parâmetros e bibliotecas empregadas, o processo de predição dos modelos, as métricas de avaliação adotadas, a implementação do sintetizador de voz e as tecnologias aplicadas. Já em Capítulo 4, são apresentados e analisados os desempenhos dos modelos convolucional, FFCNN e VGG16, além do reconhecimento em tempo real utilizando a *webcam*. Por fim, o Capítulo 5 resume as principais descobertas e implicações da pesquisa, destacando os resultados obtidos e sugerindo direções para futuros trabalhos.

2 REFERENCIAL TEÓRICO

Nas seções a seguir serão apresentados os elementos teóricos necessários para embasar a execução de um modelo computacional que utiliza visão computacional e redes neurais para tradução de libras e sintetizador de voz utilizando *Python*.

2.1 LÍNGUA BRASILEIRA DE SINAIS

A Libras (Língua Brasileira de Sinais) foi oficializada como língua das pessoas surdas no Brasil pela Lei n.º 10.436/2002, que define língua como:

> Parágrafo único. Entende-se como Língua Brasileira de Sinais — Libras a forma de comunicação e expressão, em que o sistema linguístico de natureza visual-motora, com estrutura gramatical própria, constituem um sistema linguístico de transmissão de ideias e fatos, oriundos de comunidades de pessoas surdas do Brasil.

Assim como outras línguas de sinais ao redor do mundo, como a ASL (*American Sign Language*) nos Estados Unidos, a BSL (*British Sign Language*) na Inglaterra, a *Lengua Española de Signos na Espanha* e a LSF (*Langue des Signes Française*) na França, a Libras é uma língua natural, visual-motora, com estrutura gramatical própria.

As línguas de sinais são naturais e surgiram do convívio entre as pessoas surdas. Elas são comparáveis em complexidade e expressividade às línguas orais, possuindo regras e estruturação próprias, o que as torna línguas completas e não meros conjuntos de gestos (HONORA, 2020).

Cada palavra na Libras é representada por um sinal, diferenciando-se por regras gramaticais específicas. A comunicação na Libras é gestual-visual, em que as mãos emitem os sinais e os olhos do receptor os interpretam. Essa língua é direcionada não apenas para pessoas surdas, mas também para surdo-cegas e pessoas surdas que não possuem braços, adaptando-se a diferentes necessidades de comunicação.

A estrutura gramatical da Libras possui uma estrutura frasal particular, diferenciando-se da Língua Portuguesa. Enquanto o português segue a sequência sujeito-verbo-objeto na formação de frases, na Libras utiliza-se o objeto-verbo-sujeito ou objeto-sujeito-verbo, refletindo a ênfase dada ao objeto na comunicação dos surdos.

Para uma comunicação eficaz em Libras, é importante recorrer ao Alfabeto Manual em casos de desconhecimento de palavras ou frases, respeitar o espaço pessoal dos surdos, utilizar expressões faciais para dar sentido aos sinais, manter contato visual, repetir sinais para ênfase e praticar constantemente para aprimorar o aprendizado da língua.

Conforme a Figura 1, o alfabeto e os números em Libras são representados utilizando apenas uma mão, onde 20 letras e os números são representados por sinais estáticos e 6 letras são sinais dinâmicos.

Figura 1 – Alfabeto de Libras e Configuração de Mãos.

Fonte: Grupo de pesquisa do curso de LIBRAS do Instituto Nacional de Educação de Surdos.

A promoção e difusão da Libras são essenciais para garantir a comunicação e integração plenas das pessoas surdas na sociedade. Embora tenham sido feitos avanços significativos, ainda existem desafios a serem enfrentados, como a garantia de acesso a serviços e tecnologias adequadas.

2.2 MACHINE LEARNING

A IA (Inteligência Artificial) permite que computadores e máquinas realizem tarefas que simulem o raciocínio humano, no que tange a resolução de problemas e tomadas de decisão, por meio de algoritmos inteligentes. A IA possui uma subárea denominada de ML (*Machine Learning*) ou Aprendizado de máquina, que estuda o aprendizado de sistemas inteligentes com objetivo de compreender e reconhecer padrões para realizar tomadas de decisões (BODEN, 1996).

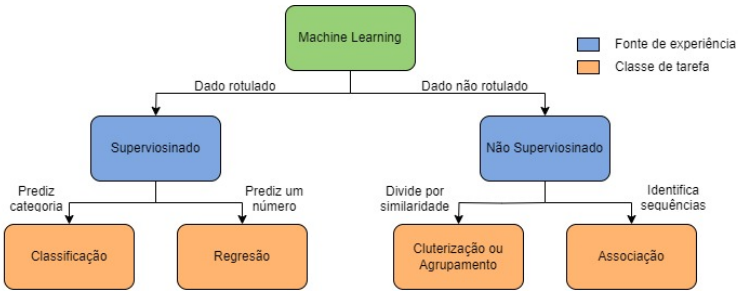
Conforme a definição de aprendizado de máquina por (MITCHELL, 1997).

> Diz-se que um programa de computador aprende com a experiência E com respeito a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho nas tarefas em T, conforme medido por P, melhora com a experiência.

Por exemplo, consideremos a tarefa T de classificar imagens de cães e gatos, utilizando como experiência E um conjunto de dados, previamente conhecido, contendo imagens de ambos os animais. A medida de desempenho P deste programa pode ser a acurácia, a qual se aprimora conforme aumenta a experiência, isto é, à medida que mais classificações são realizadas.

Dentre alguns tipos de tarefa que um sistema de aprendizado adota, a classificação é bastante utilizada, esta tarefa reconhece padrões nos dados utilizados para o treinamento dos algoritmos. (MITCHELL, 1997). Os programas que realizam esse tipo de tarefa recebem dados de entrada e atribuem rótulos específicos (classes) a esses dados com base na experiência. Essa experiência está relacionada ao conjunto de dados utilizado no processo de aprendizagem. Se o conjunto de dados for rotulado, o aprendizado é supervisionado. Se não for rotulado, o aprendizado é não supervisionado. Portanto, grande parte dos algoritmos de aprendizado clássico são categorizados conforme mostra a Figura 2.

Figura 2 – Aprendizado de Máquina Clássico.



Fonte: Próprio Autor (2024).

É necessária uma medida quantitativa que determine se o algoritmo está realizando sua tarefa de maneira satisfatória, logo, conforme o objetivo, existem medidas de desempenho especificas para a tarefa. Por exemplo, para algoritmos do tipo classificação as mais comuns são: acurácia, proporção de predições corretas em relação ao total de predições feitas pelo modelo; precisão, proporção de exemplos positivos corretamente classificados em relação ao total de exemplos classificados como positivos e Matriz de Confusão, consiste em uma tabela que mostra a frequência com que cada classe é classificada corretamente ou incorretamente.

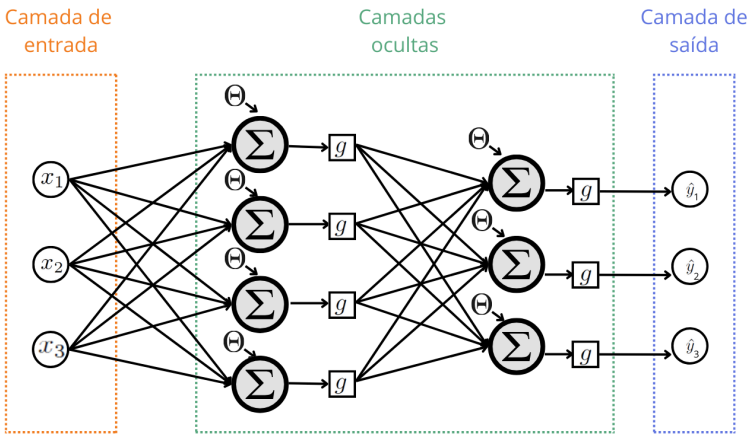
2.3 REDES NEURAIS

A Rede Neural (RN), é uma das técnicas de aprendizado de máquina, consiste em um método de inteligência artificial inspirado no funcionamento do cérebro humano, que basicamente ensina computadores a processar dados. Quando aplicadas em visão computacional, contribuem para avanços significativos, permitindo a identificação de objetos, reconhecimento de rostos e interpretação de cenas.

Uma RN básica possui neurônios interconectados em três camadas: entrada, oculta e saída. Na camada de entrada, os dados externos são processados e encaminhados para a próxima camada. As camadas ocultas recebem entradas das camadas anteriores, analisam e processam esses dados antes de enviá-los adiante. Por fim, a camada de saída fornece o

resultado do processamento, podendo ter um ou vários nós dependendo do problema a ser resolvido. Já as redes neurais profundas possuem múltiplas camadas ocultas, com inúmeros neurônios interligados por conexões ponderadas, conhecidas como pesos (BRAGA, 2007). Essas redes têm potencial para lidar com uma ampla gama de entradas e saídas, mas requerem extenso treinamento com grandes conjuntos de dados para funcionar eficazmente.

Figura 3 – Rede Neural Artificial.



Fonte: Curso de Machine Learning - 7 Redes Neurais.

Matematicamente, uma RN é uma composição de transformações lineares e não lineares. A equação (1) é o modelo matemático para uma RN onde a saída de um determinado neurônio (k) é representada por y_k, sendo a resultante de uma somatória do vetor de entrada (x_j) multiplicado pelos respectivos pesos (w_{kj}), gerando um produto interno, e tudo isso somado ao bias (viés) (Θ_k), o qual aumenta ou diminui a entrada da função de ativação (g) (FLECK *et al.*, 2016), que serão detalhadas na subseção 2.3.1.

$$y_k = g\left[(\sum_{j=1}^{M} x_j w_{kj}) + \Theta_k\right] \tag{1}$$

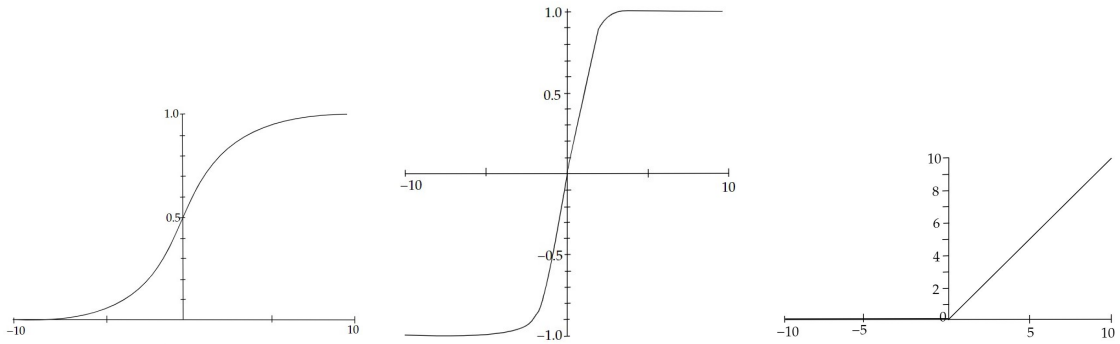
Durante o treinamento supervisionado, o algoritmo de retro propagação é essencial. Funciona em duas etapas principais. Primeiro, na propagação para frente, os dados de entrada passam pela rede camada por camada até gerar uma saída. Em seguida, calcula-se o erro, sendo a diferença entre a saída obtida e a desejada. Na segunda etapa, de propagação para trás, esse erro é utilizado para ajustar os pesos das conexões, de modo a minimizar o erro. Este processo de ajustes é repetido várias vezes até que o erro seja suficientemente pequeno, permitindo que a rede aprenda a fazer previsões mais precisas.

2.3.1 Funções de Ativação

As funções de ativação são fundamentais em redes neurais por introduzirem não-linearidades, permitindo que as redes aprendam padrões complexos. A função *sigmóide*

(Figura 5(a)), representada por uma curva em forma de "S", é usada para prever probabilidades, pois seus valores variam entre 0 e 1. A tangente hiperbólica (Figura 5(b)) é semelhante à *sigmóide*, mas varia entre 1 e 1, sendo simétrica em relação à origem. A ReLU ou Linear Retificada (Figura 5(c)) se tornou popular por permitir um treinamento mais rápido e eficiente de redes profundas, especialmente em grandes conjuntos de dados. Ela ativa apenas os neurônios com entradas positivas, definindo as entradas negativas como zero, resultando em uma rede esparsa e eficiente (BRAGA, 2007).

Figura 4 – Principais funções de ativação.



(a) Sigmóide. (b) Tangente Hiperbólica. (c) Linear Retificada.

Fonte: Próprio Autor (2024)

Além das funções de ativação mencionadas, há uma variedade de outras que são variantes das apresentadas, cada uma com suas próprias características e vantagens específicas. Por exemplo, a *Leaky ReLU* é uma modificação da ReLU que permite pequenas entradas negativas, prevenindo a inatividade de neurônios e mitigando o problema de "neurônios mortos". A ELU (*Exponential Linear Unit*) combina as vantagens da ReLU com uma curva mais suave para entradas negativas, o que pode acelerar o treinamento e melhorar a robustez da rede. A RReLU (*Randomized Leaky ReLU*) é uma variante que introduz aleatoriedade no fator de inclinação para entradas negativas durante o treinamento, promovendo maior regularização e generalização (BRAGA, 2007).

Outra função de ativação fundamental é a *Softmax*, comumente utilizada nas camadas totalmente conectadas das redes neurais, particularmente na camada de saída, por ser uma função de classificação eficiente. Sua principal vantagem é a capacidade de converter os valores de saída em probabilidades, variando entre 0 e 1. Por exemplo, para uma saída como [2.4, 1.8, 1.5], ao aplicar a função *Softmax*, obteríamos [0.42, 0.31, 0.27], representando a probabilidade de cada classe. Isso facilita a identificação da classe à qual a imagem de entrada pertence, tornando a *Softmax* essencial para tarefas de classificação.

2.3.2 Redes Neurais Convolucionais

Uma categoria das RNs são as redes neurais convolucionais (CNN), destacam-se por sua eficácia no processamento de entradas de imagem, voz ou áudio. Conforme a

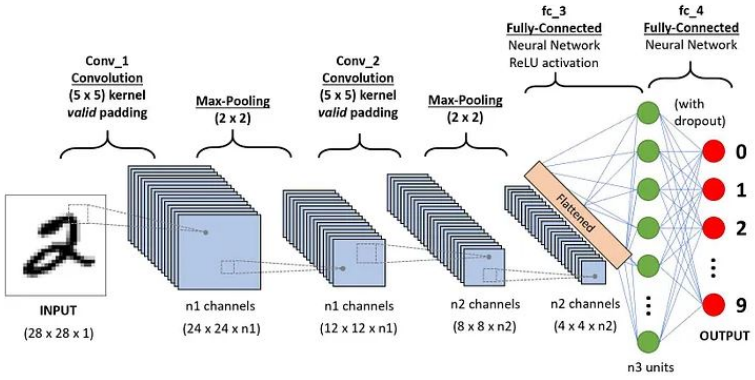
Figura 5, as principais estruturas de uma CNN são:

1. **Camada de Entrada (Input):** A camada de entrada de uma CNN recebe e prepara imagens para processamento. Podendo ajustar contraste e redimensionar imagens, por exemplo, para um formato 100×100×3, indicando 100×100 *pixels* e 3 canais de cor (LECUN; KAVUKCUOGLU; FARABET, 2010).
2. **Camada de Convolução (*convolutions*):** Na camada convolucional, os cálculos são realizados, sendo a parte central da CNN. São aplicados filtros em toda a matriz de *pixels* da imagem, incluindo cada canal RGB (*Red-Green-Blue* ou Vermelho-Verde-Azul) em imagens coloridas. A convolução é realizada movendo o filtro pela imagem e multiplicando e somando os valores correspondentes. Após a convolução, cada *pixel* segue uma função de ativação, comumente utiliza-se a ReLU, que atribui zero a resultados negativos e mantém os positivos. Isso permite identificar características na imagem, como ausência de cores (preto) (LECUN; KAVUKCUOGLU; FARABET, 2010).
3. **Camada de Agrupamento (*subsampling/pooling*):** Reduz a dimensionalidade, preservando características essenciais. Ela usa uma função de agregação para selecionar valores significativos da entrada, exemplo a operação "*max pooling*" que mantém apenas o valor máximo em uma região, para armazenar apenas as características mais proeminentes (LECUN; KAVUKCUOGLU; FARABET, 2010).
4. **Camada de Achatamanento (*flatten*):** É um processo necessário para a camada posterior, é responsável por converter os dados em matrizes unidimensionais (LECUN; KAVUKCUOGLU; FARABET, 2010).
5. **Camada Totalmente Conectada:** Realiza a classificação com base nas características extraídas anteriormente, conectando diretamente cada nó à camada anterior. Essa camada pode usar funções de ativação conforme o tipo de tarefa. Por exemplo, para tarefas de classificação a *softmax* é muito utilizada, para produzir uma probabilidade de classificação. Ao passar pelas camadas, a CNN identifica elementos maiores da imagem, criando uma hierarquia de características que culmina na identificação do objeto desejado (LECUN; KAVUKCUOGLU; FARABET, 2010).

2.3.2.1 Convoluções

A camada de convolução de uma rede neural aplica filtros, ou kernels, à entrada para extrair características específicas. Esses filtros, cujos pesos são inicialmente aleatórios e ajustados durante o treinamento, são aplicados a pequenas regiões da entrada chamadas de *receptive fields*(NIELSEN, 2019). A profundidade da saída da convolução é determinada pela quantidade de filtros utilizados, e camadas mais profundas identificam traços mais

Figura 5 – Redes Neurais Convolucionais.



Fonte: O Poder das Redes Neurais Convolucionais.

detalhados. Além dos tamanhos dos filtros, a convolução é definida por parâmetros como o *stride*, que determina o passo do filtro sobre a entrada, e o *padding*, que pode ser adicionado para evitar a rápida redução do tamanho das camadas e preservar mais informações na borda da entrada. Para exemplificar este processo a Figura 6, aplica um filtro, que é uma matriz bidimensional de pesos, sobre a imagem. Normalmente, esse filtro é uma matriz 3x3, que determina o tamanho do campo receptivo. O filtro é aplicado a uma área da imagem, onde é calculado um produto escalar (*dot-product*), entre os *pixels* da imagem e os pesos do filtro.

O *dot-product* (produto escalar) é uma operação que multiplica correspondentes elementos de duas matrizes e soma esses produtos. No contexto da convolução, isso significa que cada valor do filtro é multiplicado pelo correspondente valor da área da imagem que o filtro está cobrindo, e então todos esses produtos são somados para formar um único valor de saída.

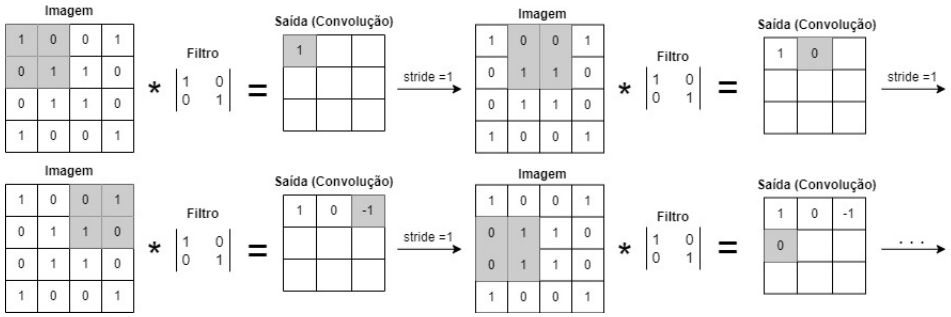
Essa operação é repetida conforme o filtro se desloca por um avanço, cobrindo toda a imagem. A saída final, composta pelos resultados dos *dot-products* da imagem e do filtro, é conhecida como mapa de feição, mapa de ativação ou feição convolvida. O *dot-product* é utilizado porque ele permite capturar a semelhança local entre o filtro e a imagem, o que é essencial para a detecção de características na imagem.

2.3.3 FFCNN (*Feature Fusion-based Convolutional Neural Network*))

Proposta por (CHEVTCHENKO *et al.*, 2018), a FFCNN íntegra os resultados de diferentes aplicações de filtros em uma imagem em uma única CNN. Em vez de criar uma rede com múltiplas camadas, a FFCNN permite criar duas redes com menos camadas e juntá-las posteriormente, reduzindo a quantidade de parâmetros a serem calculados. A FFCNN baseia-se em três pilares principais:

1. **Rede CNN Tradicional:** Uma CNN tradicional que recebe uma imagem de

Figura 6 – Exemplo de camada convolucional



Fonte: Próprio Autor (2024)

32x32 *pixels*, composta por duas camadas de convolução seguidas de funções de ativação, e duas camadas de max pooling.

2. **CNN com Filtro Gabor:** Uma CNN similar que recebe a imagem após a aplicação de um filtro Gabor (KAMARAINEN; KYRKI; KALVIAINEN, 2006), que elimina ruídos por meio de equações senoides, sendo útil em tarefas de segmentação, como em impressões digitais.
3. **Filtragem do Contorno com Momentos de Zernike:** O contorno do gesto é filtrado usando os momentos de Zernike, que extraem os contornos da mão e os inserem como entrada na rede densamente conectada que executa a classificação da imagem.

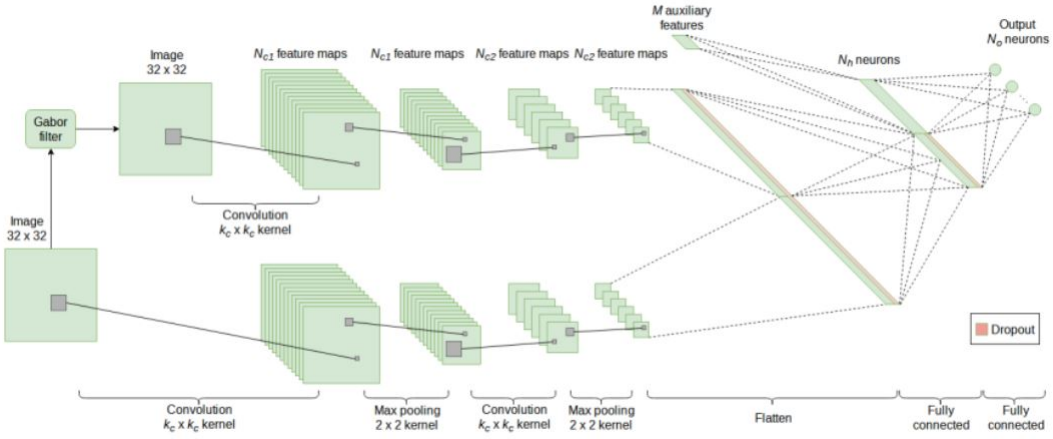
Assim, a FFCNN utiliza três fontes de dados: a imagem original, a imagem filtrada por Gabor e os contornos filtrados pelos momentos de Zernike. Essas entradas são combinadas em uma rede densamente conectada, que processa e classifica a imagem em diversos atributos de características. Conforme mostra a Figura 7.

2.3.4 Redes VGG (*Visual Geometry Group*)

A VGG é uma arquitetura de rede neural convolucional clássica, introduzida por Simonyan e Zisserman em 2014, conhecida por sua simplicidade e eficácia. Baseada na ideia de aumentar a profundidade das redes anteriores, a VGG processa imagens por meio de uma pilha de camadas de convolução seguidas de funções de ativação ReLU. Utiliza filtros pequenos de 3×3 *pixels*, com um passo (*stride*) de 1 *pixel*, preservando a resolução espacial da imagem após cada convolução. Cada bloco de convolução é seguido por uma camada de max pooling com uma janela de 2×2, que reduz a dimensionalidade e destaca as características mais importantes.

Conforme mostra a Figura 8, após passar por várias camadas de convolução e *pooling*, os dados são direcionados para uma rede totalmente conectada (MLP) composta por três camadas: duas com 4096 neurônios e uma com 1000 neurônios, correspondendo

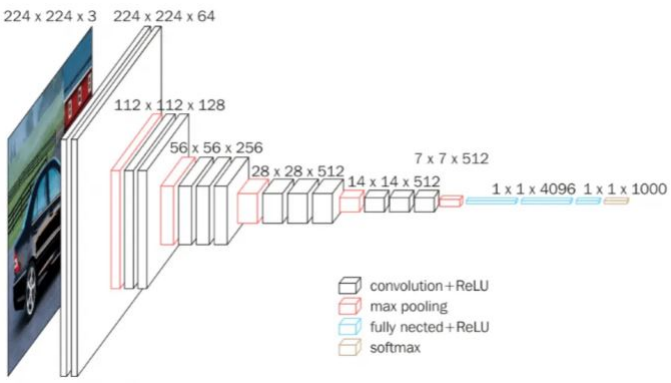
Figura 7 – Exemplo de rede FFCNN.



Fonte: (CHEVTCHENKO *et al.*, 2018)

ao número de classes no problema de classificação. Cada uma dessas camadas totalmente conectadas utiliza a função de ativação ReLU, e a última camada aplica a função *softmax* para produzir probabilidades de classificação.

Figura 8 – Exemplo de arquitetura de rede VGG.



Fonte: (LUCAS, 2021)

Embora eficaz, o treinamento da VGG é extremamente lento devido à sua profundidade e às camadas densamente conectadas. Estudos mostraram (MEDIUM, 2021) que treinar uma VGG do zero pode levar aproximadamente 10 dias usando oito GPUs, o que reflete a complexidade e os recursos computacionais necessários para essa arquitetura.

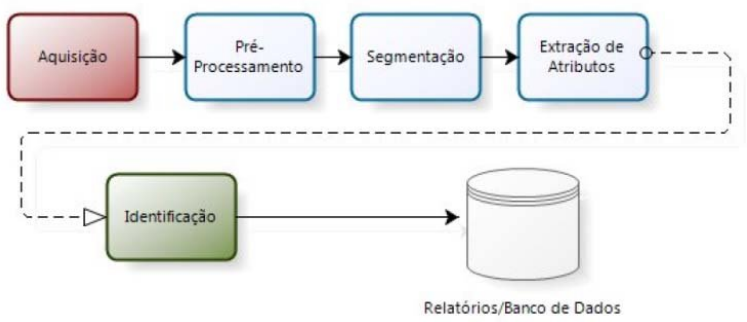
2.4 VISÃO COMPUTACIONAL

Visão computacional é a ciência que estuda e desenvolve tecnologias possibilitando que as máquinas extraiam características do ambiente, através da análise de imagens

capturadas por uma variedade de dispositivos, como câmeras de vídeo e sensores. As informações obtidas permitem reconhecer, manipular e processar dados sobre os objetos que compõem a imagem capturada. (BALLARD; BROWN, 1982).

As aplicações de sistema de visão computacional abrangem diversas áreas como reconhecimento facial, segurança, medicina, industrial entre outros. Na maioria das aplicações o sistema possui as etapas da Figura 9.

Figura 9 – Etapas de um sistema de Visão Computacional Típico.



Fonte: (RESEARCHGATE, 2015).

O sistema de visão computacional se inicia pela aquisição de imagem, onde sensores digitais capturam e digitalizam a imagem do ambiente. O processo de captura de imagens digitais começa pelos capacitores fotossensíveis presentes nos sensores, que produzem um sinal elétrico proporcional a carga de energia correspondente a intensidade luminosa, em seguida um digitalizador, converte o sinal elétrico analógico em um sinal digital (ALBUQUERQUE *et al.*, 2012).

Posteriormente, essa imagem digitalizada passa por um pré-processamento, cujo objetivo é corrigir eventuais ruídos e distorções presentes na imagem. Nessa fase, são aplicados filtros, representados por matrizes de diferentes tamanhos, que percorrem a imagem destacando o objeto de interesse. Esses filtros têm a função de realçar regiões específicas da imagem, como bordas e formas geométricas, tornando-as mais evidentes para a identificação e análise subsequentes (BARELLI, 2018).

A terceira etapa, consiste em segmentar os objetos de interesse, a fim de obter características desses objetos. A segmentação de imagens trata de duas propriedades fundamentais, a primeira é a descontinuidade, visa separar a imagem com base em variações abruptas de níveis de cinza, como bordas. A segunda, é a similaridade, processo onde se agrupam pontos da imagem com valores semelhantes para determinadas características, como na limiarização, onde um limiar de intensidade é definido para criar uma imagem binária. *Pixels* acima do limiar são destacados como primeiro plano, enquanto os abaixo são considerados segundo plano (PEDRINI; SCHWARTZ, 2008).

Com as características dos objetos, obtidas na etapa de segmentação, a etapa

de extração de atributos identifica e extrai parâmetros quantitativos ou qualitativos que possam ser usados por algoritmos de aprendizado de máquina ou outras técnicas de análise, podendo incluir identificação de características como bordas, texturas, formas geométricas, cores predominantes, entre outros elementos distintivos presentes nas imagens (CONCI; AZEVEDO; LETA, 2008).

Por fim, na etapa de processamento de alto nível ocorre a identificação. Logo, o objetivo é reconhecer e classificar objetos ou padrões específicos nas imagens. Isso geralmente envolve o uso de algoritmos de aprendizado de máquina ou técnicas de processamento de imagem para comparar os atributos extraídos das imagens com padrões previamente definidos em um conjunto de dados de treinamento. Nesta etapa frequentemente utilizam-se técnicas de aprendizado de máquina, como redes neurais convolucionais (CNNs), as quais são especialmente eficazes para tarefas de classificação de imagem.

2.5 FRAMEWORKS E BIBLIOTECAS

Neste tópico serão apresentados *Frameworks* e bibliotecas em *Python*, linguagem de programação versátil e com ampla aplicabilidade, muito utilizados em projetos de visão computacional. Por fim, é apresentado a plataforma *Google Colab* que pode ser utilizada para executar as bibliotecas citadas.

2.5.1 *TensorFlow* e *Keras*

O *TensorFlow* é uma plataforma de aprendizado de máquina de código aberto desenvolvida pela Google, e o *Keras*, uma API de alto nível e ótima usabilidade lançada como parte do *TensorFlow*. É possível usar o nível de abstração fornecido pelo *Keras*, para diversas aplicações, proporcionando uma gama de funcionalidades, sem a complexidade dos conceitos de nível inferior do *TensorFlow*.

Essa ferramenta facilita a construção, treinamento e implementação de redes neurais de forma rápida e intuitiva. O *Keras* fornece as diversas funcionalidades para treinamento e predição de modelos, como funções de perda, otimizadores, métricas de avaliação, pré-processamento e aumento de dados, etc.

2.5.2 *Numpy*

Numpy é crucial para manipulação eficiente de dados numéricos em projetos de visão computacional, permitindo operações matriciais, indexação, fatiamento e estatísticas. Comumente utilizado com outras bibliotecas, como *TensorFlow* e *Keras*, para processar imagens.

2.5.3 OpenCV

OpenCV é uma biblioteca amplamente utilizada em projetos de visão computacional. Suas principais funcionalidades incluem processamento de imagem, detecção de objetos, reconhecimento de padrões e calibração de câmera. É uma ferramenta essencial para carregar, manipular e processar imagens e vídeos em aplicações de visão computacional. Sua ampla gama de funções simplifica tarefas como detecção de bordas, filtragem, transformações geométricas e segmentação de imagens. Sua integração com outras bibliotecas, como *Numpy*, facilita o processamento eficiente de dados de imagem em projetos mais complexos.

2.5.4 *Web Speech API*

Existem basicamente duas tecnologias principais para leitura e interpretação de voz (*Speech Technologies*): Reconhecimento de Fala (*Speech Recognition*, SR) e Síntese de Fala (*Speech Synthesis*). A palavra "*Speech*" refere-se aos processos associados à produção e à percepção dos sons utilizados na linguagem falada (DUTOIT, 1997).

A síntese de fala, conhecida por TTS (*Text-To-Speech*), permite que dispositivos eletrônicos leiam em voz alta qualquer texto escrito, o que é particularmente útil para acessibilidade, assistentes virtuais e aplicativos que exigem leitura automatizada. Para possibilitar este processo pode ser utilizada a *API Web Speech* que torna os aplicativos da web capazes de lidar com dados de voz, fornecendo interfaces para ambas as tecnologias de reconhecimento e síntese de fala. A API é composta por duas principais interfaces: *SpeechRecognition* e *SpeechSynthesis*. A primeira costuma ser usada em aplicativos que exigem interação por voz, como assistentes virtuais, ditado de texto e comandos de voz. Já as *SpeechSynthesis*, permite aos programas ler um conteúdo de texto por meio do sintetizador de fala padrão do dispositivo, tecnologia esta que é comumente utilizada para leitores de tela, auxiliando pessoas com limitações visuais.

A *SpeechSynthesisUtterance* é um componente essencial da *SpeechSynthesis*. Esta interface representa uma solicitação de síntese de fala e permite que desenvolvedores especifiquem texto, idioma, tom e volume. Essa interface possibilita um controle detalhado sobre como o texto será transformado em fala, permitindo ajustes para tornar a síntese mais natural e adequada ao contexto do aplicativo.

2.5.5 Google Colab

Google Colab é uma plataforma em nuvem que permite escrever e executar código *Python* diretamente no navegador, sem necessidade de configuração. Ele oferece acesso gratuito a recursos de computação, incluindo GPU e TPU, o que o torna uma escolha popular para desenvolvimento e treinamento de modelos de aprendizado de máquina.

2.5.6 *MediaPipe*

MediaPipe é uma estrutura de código aberto desenvolvida pelo *Google*, projetada para construir cadeias de processamento de mídia em tempo real. Essa biblioteca eficiente e flexível é amplamente utilizada no desenvolvimento de aplicações de visão computacional, reconhecimento de gestos, detecção e rastreamento de objetos, entre outras tarefas relacionadas ao processamento de mídia em tempo real. A estrutura é adaptável a diversas aplicações, devido a uma variedade de módulos pré-construídos que podem ser facilmente integrados e configurados para criar fluxos de processamento personalizados.

O *MediaPipe* inclui uma ferramenta chamada *Hand Landmarker*, que utiliza uma CNN treinada especificamente para detectar mãos em imagens ou quadros de vídeo. A CNN identifica regiões da imagem que contêm mãos e fornece pontos de referência detalhados que descrevem a posição e a forma da mão. A ferramenta pode ser configurada para detectar mãos em modo estático ou em vídeo contínuo, ajustar a quantidade de mãos a serem detectadas, definir a confiança mínima para a detecção ser considerada válida, entre outras opções de configuração.

2.6 TRABALHOS CORRELATOS

O escopo da pesquisa (SILVA, K. P. d., 2023) envolve o estudo das redes neurais LSTM e a tecnologia *MediaPipe*. No estudo foi realizado um treinamento com conjuntos de dados, onde modificações foram realizadas por meio de técnicas como espelhamento horizontal, translação e aumento do brilho nas sequências de imagens. De modo a avaliar o conjunto que possui melhor eficácia e eficiência, utilizando métricas de desempenho, apresentou-se um conjunto com 100% de acurácia tanto no treinamento quanto na validação.

No trabalho (SILVA, R. P. d. *et al.*, 2022) foi utilizado o framework *MediaPipe* para capturar expressões faciais, pontos de articulação, configuração, movimentação e orientação das mãos em vídeos. A ferramenta demonstra eficácia na identificação e interpretação dos sinais de teste, 4 palavras e 5 letras, destacando-se pela capacidade de analisar individualmente cada parâmetro do sinal antes de unificá-los para determinar o significado final.

Já (SANTOS ANJO; PIZZOLATO; FEUERSTACK, 2012) apresentou o sistema Gesture User Interface (GestureUI) para o reconhecimento em tempo real de gestos estáticos de Libras, utilizando um fluxo contínuo de vídeo e informações de profundidade do *Kinect*. O sistema destaca-se por permitir a integração com diferentes interfaces por meio de um protocolo TCP/IP. O reconhecimento de gestos ocorre em duas etapas principais: segmentação, aprimorada por uma Parede Virtual e heurísticas específicas para Libras; e Classificação, utilizando uma Rede Neural de Perceptrons de Múltiplas Camadas treinada pelo sistema.

A pesquisa de (CRUZ, 2019) visa identificar e classificar sinais estáticos da Língua Brasileira de Sinais (Libras). Inspirado na classificação da Língua de Sinais Americana, o estudo aplica a FFCNN aos gestos estáticos da Libras. Foram comparados três classificadores em um *dataset* de Libras com 9.600 imagens de 40 sinais, sendo a rede VGG com 16 camadas a mais precisa, alcançando 99,45% de acurácia, porém, obteve uma acurácia menor em relação à CNN, com a taxa de aprendizagem mais alta, devido à menor complexidade deste algoritmo.

Saindo do escopo de visão computacional e redes neurais, foram analisados projetos que envolvem sintetizadores de voz. Como o (SANTOS; DORNELLES, 2008) que foca na conversão de documentos em formato HTML (HyperText Markup Language) para áudio. O sistema desenvolvido, utilizando o *Framework .NET*, incorpora uma biblioteca denominada YETI para a conversão em formato mp3, além da API SAPI para a síntese de voz utilizando SSML. O trabalho visa aprimorar a tecnologia TTS com características específicas, como o tratamento de nuances na leitura de diferentes trechos de texto (parágrafos, fim de página, cabeçalho, etc.), a fim de aprofundar a eficácia do processo de sintetização. O sistema oferece opções como salvar arquivos em formatos de áudio (mp3 e *wave*), além de proporcionar controle sobre volume e velocidade.

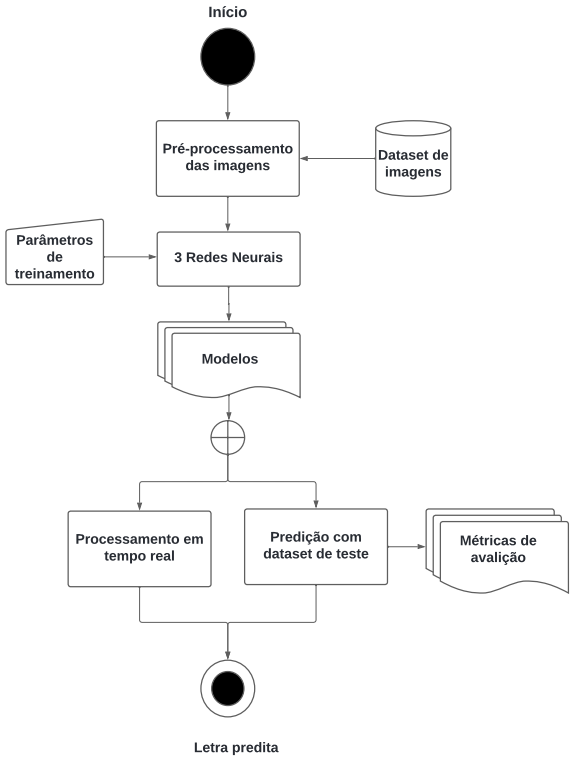
A revisão dos trabalhos relacionados revelou diversas abordagens no reconhecimento e tradução de gestos de Libras, empregando tecnologias como redes neurais LSTM e FFCNN, *MediaPipe*, *GestureUI* e TTS. No entanto, uma limitação significativa é que as pesquisas são majoritariamente conduzidas por pesquisadores brasileiros, o que pode restringir a diversidade de métodos e técnicas exploradas. Além disso, os trabalhos existentes muitas vezes focam em subconjuntos limitados de sinais ou utilizam *datasets* específicos, não abrangendo todas as letras do alfabeto ou diferentes contextos de uso. O presente estudo se diferencia ao integrar um modelo de visão computacional com redes neurais convolucionais (CNN, FFCNN, VGG16) que alcançam altos índices de acuracidade, para tradução em tempo real de letras de Libras, utilizando um sintetizador de voz, e implementando tudo em uma plataforma em nuvem, facilitando a execução em máquinas de baixa desempenho. Essa abordagem visa superar algumas das limitações anteriores, oferecendo uma solução mais acessível e abrangente para o reconhecimento de gestos em Libras.

Portanto, foi apresentado acima, os conceitos e técnicas essenciais para o desenvolvimento desta pesquisa, bem como as tecnologias e frameworks utilizados. Esta fundamentação teórica proporciona o avanço para próxima seção, onde será discutido o desenvolvimento e implementação do modelo computacional.

3 IMPLEMENTAÇÃO DE UM MODELO COMPUTACIONAL DE RECONHECIMENTO DE SINAIS DE LIBRAS

O desenvolvimento deste trabalho seguiu o fluxograma apresentado na Figura 10, e nesta seção é detalhado a metodologia adotada, dividida em etapas específicas: base de dados, pré-processamento, treinamento de redes neurais, métricas de avaliação, sintetizador de voz e tecnologias utilizadas.

Figura 10 – Fluxograma Geral de Desenvolvimento



Fonte: Próprio Autor (2024)

3.1 BASE DE DADOS (**DATASET**)

Para a realização deste trabalho, foi essencial a seleção de um conjunto de dados apropriado que representasse adequadamente o alfabeto manual da Língua Brasileira de Sinais (Libras). Optou-se por utilizar uma base de dados disponível em um repositório público no GitHub (MARQUES, 2022), especificamente criada para gestos estáticos do alfabeto manual de Libras. Logo, das 26 letras do alfabeto, apenas 21 estão presentes no conjunto de dados. Importante dizer, que a escolha desta base de dados se dá, pela sua abrangência e disponibilidade para fins de pesquisa.

O conjunto de dados é composto por um total de 46.262 amostras de imagens, de maneira aleatória. As imagens são subdivididas em pastas de: treino, validação e

teste, contendo 32.379 (aproximadamente 70%), 4.622 (10%) e 9.261 (20%) amostras, respectivamente. Cada imagem é categorizada por meio de subpastas, que indicam o rótulo correspondente a cada letra do alfabeto.

A organização do conjunto de dados desta maneira foi escolhida estrategicamente visando a facilidade de uso em determinadas redes neurais, como a CNN, uma vez que esta arquitetura é amplamente utilizada em tarefas de classificação de imagens. A estrutura de diretórios permite que o conjunto de dados seja carregado com o gerador de imagens fornecido pelo Keras. O gerador, é uma ferramenta que carrega e processa imagens em tempo real durante o treinamento de modelos. Eles permitem aplicar aumentos (augmentations) e normalizações automaticamente, facilitando a gestão de grandes conjuntos de dados e melhorando a generalização do modelo.

3.2 PRÉ-PROCESSAMENTO DOS DADOS

3.2.1 Pré-processamento dos dados para treinamento do modelo

A etapa de pré-processamento do conjunto de dados desempenha um papel crucial no aprendizado de máquina, sendo indispensável para aprimorar a eficácia do treinamento das técnicas de aprendizado. O uso de Redes Neurais Profundas combinadas com filtros de convolução para a extração de recursos, resultam em uma abordagem moderna de aprendizado de máquina. Dessa forma, procedimentos como redução de ruído, aumento de contraste, tornam-se desnecessários, por não precisar gerar os recursos (*features*) manualmente.

Portanto, optou-se por apenas normalizar os valores de *pixel* das imagens, para ajustar a faixa de valores de 0-255 para 0-1. Essa normalização é uma prática comum em aprendizado de máquina por ajudar a estabilizar o treinamento da rede neural, facilitando a convergência do modelo.

3.2.2 Pré-processamento dos dados para predição em tempo real

O pré-processamento das imagens de entrada para o modelo de tradução com *webcam* em tempo real, começa com a conversão da imagem de saída da função em JavaScript, utilizada para captura dos *frames* da *webcam*, para um formato utilizável pela biblioteca *OpenCV*. Em seguida, a imagem pode ser espelhada horizontalmente para corrigir a orientação. A conversão de cores foi realizada para transformar a imagem do espaço de cor BGR para RGB, este processo se faz necessário, pois o *OpenCV* retorna as imagens com os canais de cores invertidos, isto é, em formato BGR, e o modelo foi treinado com imagens RGB.

Após isso, uma região de interesse correspondente à mão foi recortada da imagem original com base em coordenadas especificadas, conforme explicado no Tópico 3.4. A imagem da mão recortada foi então redimensionada para um tamanho padrão definido,

garantindo o tamanho apropriado para entrada do modelo. Posteriormente, os valores dos *pixels* da imagem foram normalizados para a faixa de 0-1. Finalmente, a imagem foi expandida para incluir uma dimensão adicional, preparando-a como um lote de tamanho 1 para ser diretamente alimentada ao modelo.

3.3 REDES NEURAIS

Analisando as acurácias de redes neurais para reconhecimento de sinais estáticos (CHEVTCHENKO *et al.*, 2018), foram selecionadas as redes CNN, VGG16 e FFCNN. Portanto, para treinamento das redes neurais, foram utilizadas ferramentas da biblioteca do Keras, melhor apresentada na Seção 2.5.1. Foram treinadas com o mesmo *dataset* e pré-processamento de dados.

3.3.1 Arquitetura CNN

Cada rede pode ter a referência da arquitetura das camadas, como exemplificado na Figura 20(a), que mostra as camadas aplicadas em uma arquitetura CNN. A arquitetura e os parâmetros, apresentados no próximo tópico, da CNN foram definidos com estrutura similar à arquitetura LeNet, demonstrada na Figura 5, proposta por LeCun, incluindo algumas alterações conforme a experimentação e com auxílio de projetos de código aberto (LACERDA, 2019). Embora baseada na LeNet, essa rede foi criada do zero e não utilizou *transfer learning*, uma técnica que aproveita modelos pré-treinados em grandes conjuntos de dados para melhorar o desempenho em uma nova tarefa. Nesse caso, não se fez necessário para obter bons resultados, porém, para as outras RNs, optou-se por utilizar. A rede resultante é sequencial, ou seja, as camadas da rede são adicionadas uma após a outra. A arquitetura possui oito camadas, incluindo três camadas de convolução, três camadas de subamostragem (também conhecidas como *pooling*), uma camada totalmente conectada e uma camada de saída.

De maneira geral a arquitetura possui o seguinte comportamento:

1. **Primeira e segunda camada (CONV → RELU → POOL):** A primeira camada convolucional (Conv2D) tem 8 filtros (canais de saída), cada um com um kernel de tamanho 3×3. A função de ativação utilizada após a convolução é a ReLU (Rectified Linear Unit), que ajuda na introdução de não-linearidades na rede. Em seguida, é aplicada uma camada de *pooling* (*MaxPooling2D*) com uma janela de *pooling* de tamanho 2×2, que reduz a dimensionalidade da representação e torna a computação mais eficiente.
2. **Terceira e quarta camada (CONV → RELU → POOL):** A segunda camada convolucional tem 16 filtros e utiliza a mesma configuração da primeira camada.

3. **Quinta e sexta camada (CONV → RELU → POOL):** A terceira camada convolucional tem 32 filtros e segue a mesma configuração das camadas anteriores.
4. **Sétima camada (FC ou Fully Connected):** Após as camadas convolucionais e de *pooling*, a rede é "achatada" (*Flatten*) para um vetor unidimensional para ser alimentada em uma rede neural densa (*fully connected*). Uma camada densa (*Dense*) com 16 unidades e ativação ReLU é adicionada para aprender características mais complexas nos dados.
5. **Oitava camada (*Softmax*):** Finalmente, uma camada densa de saída é adicionada com um número de unidades igual ao número de classes (letras do alfabeto). A função de ativação *Softmax* é aplicada nesta camada para gerar as probabilidades de cada classe.

No total, a rede possui 24.837 parâmetros treináveis, proporcionando um bom equilíbrio entre complexidade e desempenho.

3.3.2 Arquitetura FFCNN

Conforme explicado na Seção 2.3.3, a FFCNN aplica diferentes filtros na mesma imagem e seus resultados são combinados em uma única rede.

Portanto, cada ramo de entrada corresponde a uma fonte de informação diferente: a imagem original, a imagem filtrada com o filtro Gabor e o contorno do gesto com os momentos de Zernike. Sendo que as camadas de convolução e *pooling* são separadas para cada ramo de entrada para processar as características específicas de cada tipo de informação, processo esse muito similar ao que foi feito para o treinamento da rede CNN, conforme apresentado na Seção 3.3.1.

Após a extração de características em cada ramo de entrada, os resultados são combinados em uma única rede. Por meio de uma concatenação dos resultados dos ramos de entrada em uma camada totalmente conectada para aprender a classificar os dados combinados. Na camada de saída foi aplicado a função de ativação *Softmax* para gerar as probabilidades de cada classe.

Finalizando com 15.248 parâmetros nas duas camadas de convolução e 585.421 nas três camadas densas, totalizando 600.669 parâmetros treináveis, tendo muito mais parâmetros que a CNN, pois não esta rede não compartilha pesos como as camadas convolucionais, resultando em um número muito maior de parâmetros treináveis.

3.3.3 Arquitetura VGG16

A rede VGG16 é caracterizada por sua simplicidade e profundidade, utilizando filtros convolucionais de 3×3 e camadas de *pooling* 2×2, totalizando assim 16 camadas.

As primeiras 18 camadas alternam entre convolução e *pooling*. Cada camada convolucional usa filtros 3×3 e a ativação ReLU, padronizada para que a saída da camada convolucional tenha a mesma dimensão espacial que a entrada. As camadas de *pooling* 2×2 reduzem a dimensionalidade espacial da camada anterior pela metade, ajudando a diminuir a complexidade computacional enquanto retém características essenciais.

Em seguida, as camadas de *Flatten*, transformam a matriz 3D resultante das camadas de convolução e pooling em um vetor unidimensional. Com isso é possível conectar a saída das camadas convolucionais com as camadas totalmente conectadas, compostas por duas camadas densas, cada uma com 4096 neurônios e ativação ReLU, que aprendem a mapear as características extraídas pelas camadas convolucionais para a tarefa de classificação. Por fim, a função de ativação *Softmax* gera as probabilidades de cada classe, permitindo a classificação final.

Logo a rede possui 14.714.688 parâmetros em 12 camadas de convolução e 25.214.986 nas três camadas densas, totalizando 39.929.674 parâmetros treináveis. Realizando uma comparação com as redes anteriores, esta é muito mais complexa e profunda, por possuir inúmeros filtros em cada camada convolucional e muitas conexões em camadas totalmente conectadas. Enquanto a CNN e FFCNN têm estruturas mais simples e menos parâmetros devido ao uso eficiente de convoluções e *pooling*.

3.3.4 Bibliotecas e Parâmetros

Para a implementação da rede neural, foi utilizado o módulo *Sequential Model Keras*. Este módulo permite a construção de camadas em sequência, onde cada camada possui uma matriz de entrada e uma matriz de saída. Além disso, o Keras possui uma função denominada *flow from directory*, muito útil para aplicar em bases de dados que se encontram em formato de pastas. Esta função possui parâmetros para rotular classes conforme as sub-pastas, assim como embaralhamento dos dados.

Conforme descrito na subseção 3.3.1, os parâmetros ideais foram encontrados após uma série de experimentos, sendo esses os resultantes:

- Taxa de aprendizagem: 0,0001;
- Função de ativação das camadas ocultas: ReLU;
- Função de ativação da camada de saída: *Softmax*;
- Otimizador: Adam;
- Quantidade de épocas: 100;
- Profundidade da rede: 3;
- Quantidade de filtros: 8, 16, 32;
- Tamanho do kernel: 3×3;
- Porção de treinamento: 75%;

- Porção de validação: 12,5%;
- Porção de teste: 12,5%.

3.4 PREDIÇÃO DOS MODELOS

Os três modelos (CNN, VGG16 e FFCNN) foram avaliados utilizando o *dataset* de teste, referenciado na Seção 3.1. Utilizando a função "*predict()*" da biblioteca do Keras, os modelos realizaram previsões sobre as imagens de teste, carregadas por um gerador de dados que redimensiona as imagens para o tamanho esperado pelo modelo e converte para o formato RGB. As previsões foram então comparadas com os rótulos reais para avaliar o desempenho do modelo.

Além das predições, utilizando o *dataset* de teste, foi desenvolvido um sistema para predição em tempo real, utilizando a webcam integrada a uma sintetizadora de voz.

O algoritmo inicia com a captura de vídeo da webcam, utilizando uma função em JavaScript que converte os *frames* da webcam em imagens no formato OpenCV. Cada *frame* capturado é processado para detectar e reconhecer gestos. Em seguida, utilizando a função *Hands* da estrutura do *MediaPipe*, o sistema identifica a localização das mãos na imagem e desenha retângulos ao redor das mãos detectadas, apenas para sinalizar a identificação da mão. Com a detecção das mãos as imagens são recortadas redimensionadas para o tamanho apropriado a entrada dos modelos. Em seguida a imagem é salva com fundo branco e rotacionada horizontalmente, assim como a Figura 11, podendo assim realizar a predição. A letra predita é então exibida na tela e integrada na função para sintetizar a voz para feedback em tempo real, em tempo praticamente instantâneo, garantindo uma comunicação adequada.

Figura 11 – Imagem da mão recortada e com fundo branco.

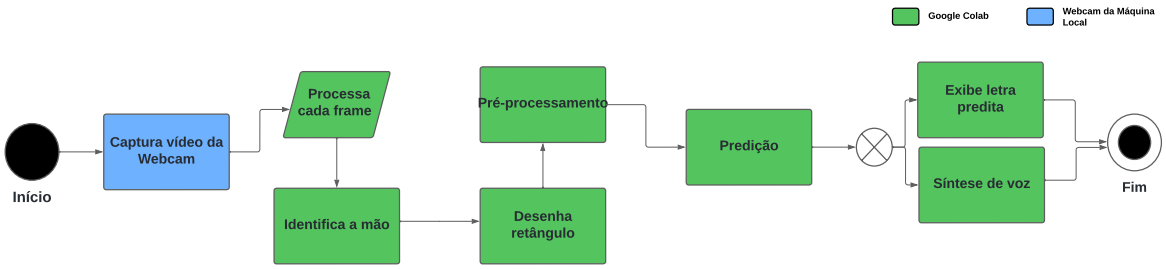
Fonte: Próprio Autor (2024).

3.5 MÉTRICAS DE AVALIAÇÃO

Com intuito de validar e analisar o modelo treinado, foram adotadas métricas de avaliação como acurácia, perda (*loss*), precisão, *recall* e *F1-Score*.

Para o cálculo dessas métricas utiliza-se os valores FP (Falso Positivo) e FN (Falso Negativo), que representam os erros de classificação. FP ocorre quando o modelo erroneamente prevê uma classe como positiva quando é negativa, enquanto FN ocorre quando

Figura 12 – Fluxograma de Reconhecimento em tempo real.



Fonte: Próprio Autor (2024).

prevê negativa quando é positiva. Além dos valores TP (Verdadeiro Positivo) e TN (Verdadeiro Negativo) indicam previsões corretas. TP é quando o modelo prevê corretamente uma classe como positiva, e TN é quando prevê corretamente como negativa. Com estes valores é possível realizar o cálculo das métricas abaixo:

- **A (Acurácia):** É uma medida simples da proporção de previsões corretas feitas pelo modelo em relação ao número total de exemplos.

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

- **loss (Perda):** Quantifica o quão bem o modelo está performando durante o treinamento, representando a diferença entre as previsões do modelo e os rótulos verdadeiros.
- **P (Precisão):** Indica a proporção de exemplos classificados corretamente como positivos em relação ao total de exemplos classificados como positivos.

$$P = \frac{TP}{TP + FP} \tag{3}$$

- **R (*Recall*):** Representa a proporção de exemplos positivos classificados corretamente como positivos em relação ao total de exemplos positivos no conjunto de dados.

$$R = \frac{TP}{TP + FN} \tag{4}$$

- **F1 (*F1-Score*):** É uma média harmônica entre a precisão e o recall, fornecendo uma única métrica que combina a precisão e a capacidade de recuperação do modelo.

$$F1 = 2.\frac{P.R}{P + R} \tag{5}$$

Outra métrica utilizada para avaliação é a matriz confusão, ferramenta útil para avaliar o desempenho de um modelo de classificação, comparando as previsões do modelo

com os valores reais. Na Figura 13, ilustra uma matriz confusão com os valores de FP, FN, TP e TN, os quais são quantificados conforme o resultado obtido ("Sim" ou "Não"). Essa análise é crucial para entender não apenas a precisão geral do modelo, mas também suas taxas de falso positivo e falso negativo, fornecendo uma visão completa de seus pontos fortes e áreas a melhorar.

Figura 13 – Matriz Confusão de 2 classes.

		Valor Predito	
		Sim	**Não**
Real	**Sim**	Verdadeiro Positivo (TP)	Falso Negativo (FN)
	Não	Falso Positivo (FP)	Verdadeiro Negativo (TN)

Fonte: Próprio Autor (2024)

As métricas citadas acima foram selecionadas por serem amplamente utilizadas em tarefas de classificação, fornecendo uma visão abrangente sobre o desempenho do modelo em termos de predições corretas, erros e equilíbrio entre precisão e sensibilidade. Embora existam outras métricas relevantes, como MAE (Erro Médio Absoluto), MSE (Erro Quadrático Médio), AUC-ROC (Área Sob a Curva), Coeficiente de Correlação de Pearson e RMSE (Raiz do Erro Quadrático Médio), estas são mais frequentemente aplicadas em tarefas de regressão ou específicas análises de correlação.

3.6 SINTETIZADOR DE VOZ

Para fornecer uma saída auditiva ao usuário, o projeto utiliza a funcionalidade de síntese de voz através da interface *SpeechSynthesisUtterance* da Web *Speech API*. Essa funcionalidade é integrada ao Ambiente do *Google Colab* por meio de uma função em JavaScript.

A chamada para o sintetizador de voz é feita diretamente no *script JavaScript* incorporado no HTML do *notebook* da aplicação local. A função *speak* é responsável por essa tarefa. Ela é chamada após o processamento da predição dos modelos. Especificamente, a função *speak* recebe o texto das letras preditas como entrada. Esse texto é passado como argumento para criar uma instância da classe *SpeechSynthesisUtterance*, que encapsula a *string* de texto que deve ser transformada em fala. O método speechSynthesis.speak() é então utilizado para converter o texto em áudio e reproduzi-lo.

Por exemplo, se os modelos preveem as letras "A", "A" e "C", a função *speak* é chamada com a *string* resultante da concatenação das predições "CNN: A, FFCNN: A, VGG16: C", conforme mostra a Figura 21. O sintetizador de voz então converte essa *string* em áudio, proporcionando um feedback auditivo ao usuário em tempo real.

3.7 TECNOLOGIAS UTILIZADAS

Durante o desenvolvimento do projeto, a linguagem de programação Python 3.10.12 foi utilizada para implementação. Para a definição da arquitetura da CNN, foram essenciais as bibliotecas *OpenCV* 4.8.0, que proporcionou funcionalidades de processamento de imagens, e o *Keras* com *TensorFlow* 2.15.0 como *backend* para a construção e treinamento do modelo.

Devido às limitações de processamento em máquina local, todos os experimentos foram conduzidos no ambiente do Google Colab. Este ambiente oferece configurações robustas, incluindo um processador Intel(R) Xeon(R) com 2 núcleos, 13GB de memória RAM e uma GPU Tesla T4, proporcionando uma plataforma de computação poderosa e eficiente para o treinamento e execução dos modelos. No Google Colab, tanto o treinamento quanto a execução dos modelos foram realizados integralmente. A captura de imagens pela webcam foi integrada diretamente no ambiente do Colab, onde as imagens capturadas são salvas no armazenamento local do Colab para uso imediato no notebook. Da mesma forma, a síntese de voz ocorre dentro do ambiente do Colab, aproveitando suas capacidades para garantir uma execução contínua e integrada de todas as etapas do processo.

3.8 CONSIDERAÇÕES FINAIS

Concluindo o capítulo de desenvolvimento, destacando a implementação das etapas do projeto, desde a captura e pré-processamento dos dados até a construção e treinamento dos modelos de redes neurais e a integração da síntese de voz. No Capítulo 4, serão apresentados os resultados obtidos a partir dos testes realizados, analisando a eficiência dos modelos implementados e discutindo suas implicações.

4 RESULTADOS

Utilizando todos os conceitos técnicos e metodologia de desenvolvimento apresentado no presente trabalho, os modelos apresentaram resultado satisfatório, fato este, comprovado pelas métricas de avaliação. Por fim, os modelos foram aplicados para reconhecimento de sinais em tempo real utilizando *webcam*, em conjunto com a ferramenta de sintetização de voz para reproduzir a letra predita.

4.1 MODELO CONVOLUCIONAL

Os resultados obtidos pela CNN, ao utilizar a arquitetura apresentada na Seção 3.3.1 e na Figura 20, foram muito satisfatórios. Na Tabela 1 é possível visualizar cada classe e seus respectivos valores de *precison*, *recall*, *f1-score* e número de amostras (*support*).

O modelo CNN demonstra resultados sólidos em suas classificações, com uma precisão média de 99%. Algumas classes alcançaram altas pontuações, todas acima de 98%. As classes com uma pontuação de recall menor em relação as demais, possuem similaridade com outros sinais, o que pode estar ocasionando os erros. Fato este que também pode ser observado na matriz confusão (Figura 14), por exemplo, a classe "D" foi predita 6 vezes como "G", como mostrado na Seção 2.1, a configuração de mãos destes sinais são parecidas. Porém, esses erros ocorreram poucas vezes em relação à quantidade de acertos, tendo pouca relevância para o desempenho do modelo, resultando em uma acurácia de 99,3%.

A Figura 15 mostra o gráfico de acurácia por época, analisando ele pode-se inferir que a acurácia aumentou ao longo das épocas, até estabilizar muito próximo de 1 após 90 épocas aproximadamente, indicando que o modelo foi melhorando sua capacidade de fazer previsões corretas, até o seu desempenho máximo. As curvas de acurácia de treino e validação estão próximas, comprovando um bom ajuste do modelo.

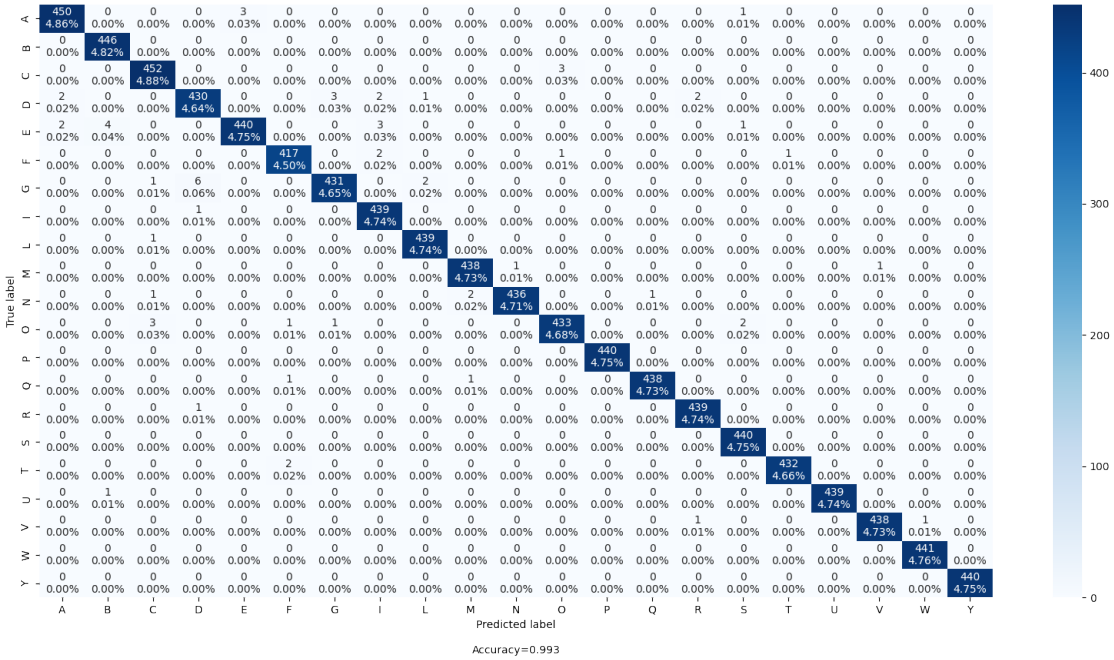
Ao analisar a perda por época, apresentado na Figura 15, a perda diminui ao longo das épocas e estabiliza próximo de 0 após 80 épocas aproximadamente, indicando que o modelo teve um ótimo ajuste aos dados de treinamento.

4.2 MODELO FFCNN

A FFCNN foi desenvolvida na arquitetura mostrada pela Figura 20, tendo um treinamento mais demorado em relação às outras redes. O modelo obteve um excelente resultado com acuracidade de 99,99%. Conforme mostra a Matiz Confusão (Figura 16) e os valores do relatório de classificação do *dataset* (Tabela 1), o modelo teve um desempenho excelente para todas as classes, atingindo uma média 100% de precisão, recall e f1-score.

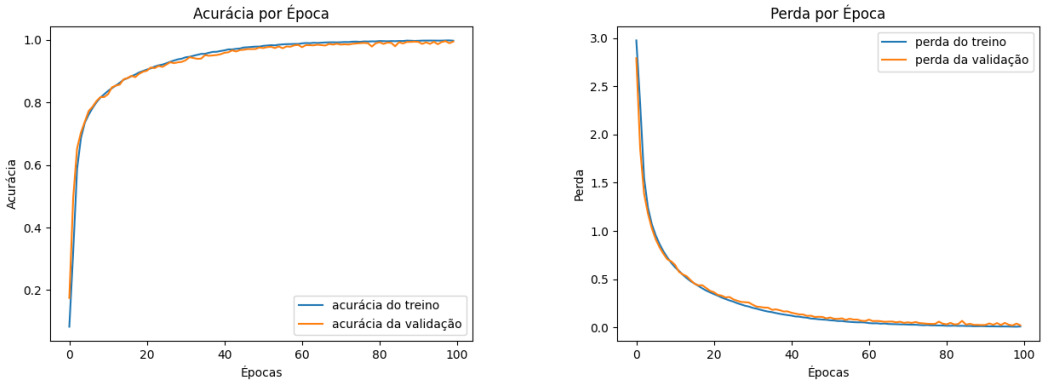
A Figura 12 mostra o gráfico de acurácia por época, este gráfico mostra um comportamento diferente em relação a outros modelos, pois geralmente a acurácia de treinamento

Figura 14 – Matriz Confusão — CNN.



Fonte: Próprio Autor (2024).

Figura 15 – Gráficos de comportamento do treinamento por época — CNN.



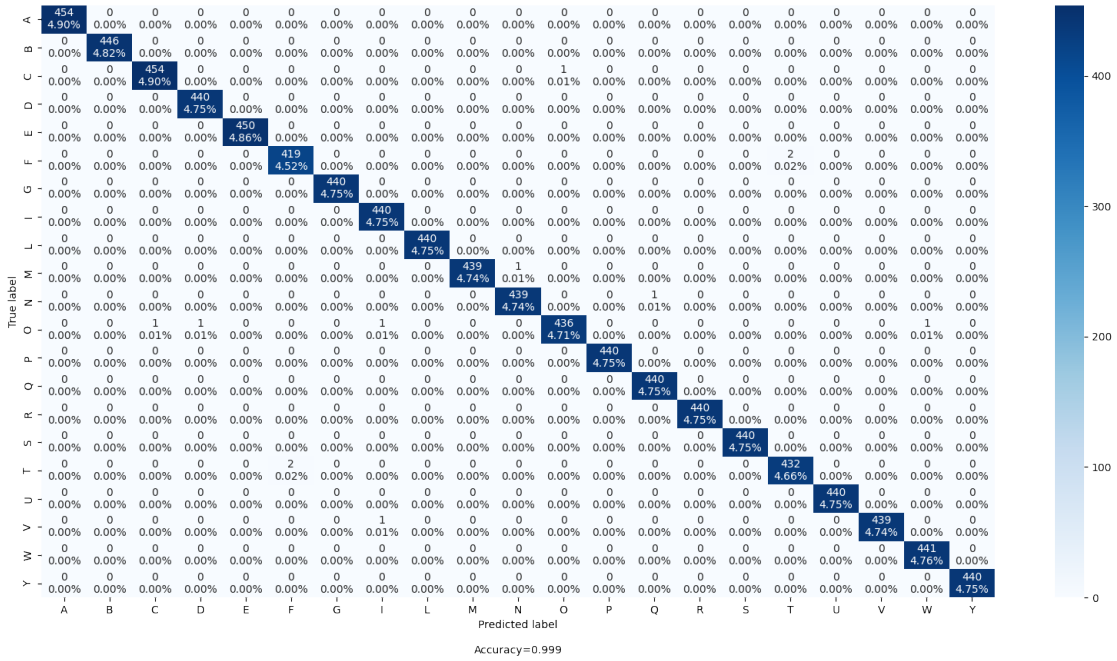
(a) Acurácia por época. (b) Perda por época.

Fonte: Próprio Autor (2024).

é maior do que a acurácia de validação, pois o modelo é otimizado para os dados de treinamento, porém neste caso a acurácia de validação foi maior, finalizando em 1 enquanto a de treino em 0,9. Essa situação não é necessariamente problemática, podendo ser causada pela regularização, distribuição dos dados e ruído do treinamento. A pequena diferença entre as duas acurácias é aceitável para o bom desempenho, além disso, ambas aumentam pregressivamente, podendo inferir que não ocorre *overfitting*.

O comportamento atípico é notável no gráfico de perda por época, Figura 17, porém, da mesma forma pode-se inferir que o modelo teve um bom comportamento, ao diminuir ao longo das épocas e estabilizar próximo de 0 após 60 épocas, aproximadamente, na validação.

Figura 16 – Matriz Confusão — FFCNN.



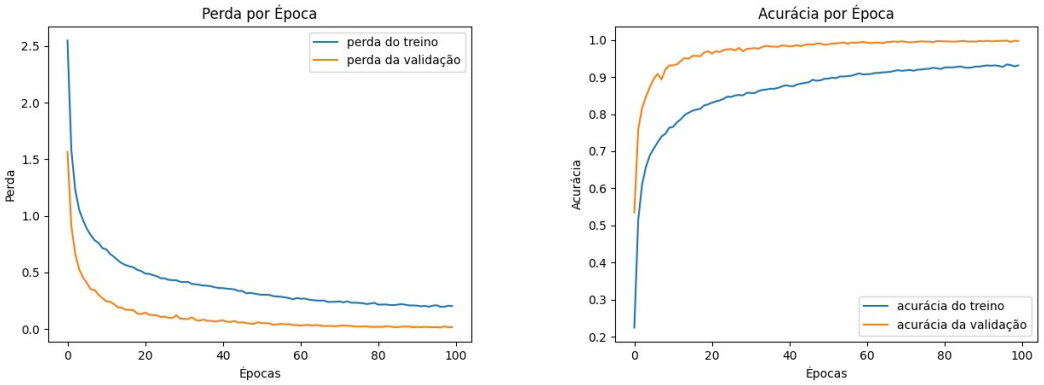
Fonte: Próprio Autor (2024).

4.3 MODELO VGG16

O modelo VGG16 com a arquitetura da Figura 20, apresentou um acuracidade de 99,8% com ótimos resultados nas classificações atingindo uma média 100% de precisão, recall e f1-score, conforme mostra a Matiz Confusão (Figura 18) e os valores do relatório de classificação do *dataset* (Tabela 1).

Conforme mostrado na Figura 19, o modelo começa com 3 de perda e mantém constante até aproximadamente a época 20, e com mais poucas épocas o modelo estabiliza em zero. Este comportamento sugere que o modelo encontrou um caminho eficiente para

Figura 17 – Gráficos de comportamento do treinamento por época — FFCNN.



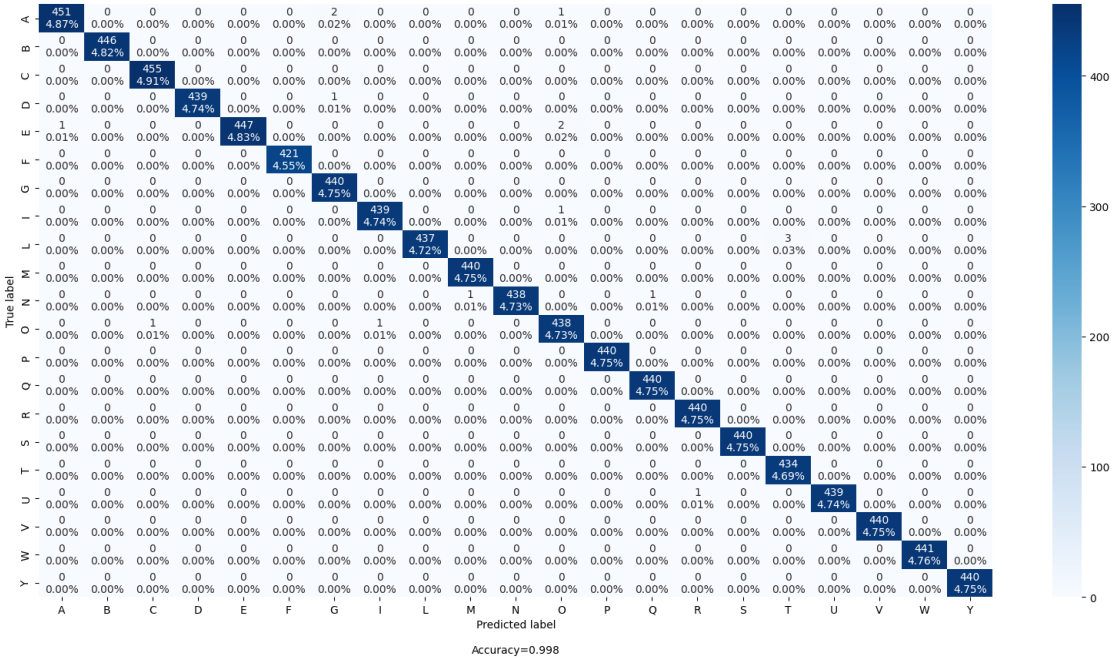
(a) Perda por época. (b) Acurácia por época.

Fonte: Próprio Autor (2024).

ajustar seus pesos após a fase inicial de exploração. Isso é comum em treinamentos onde há um ajuste significativo dos hiper-parâmetros ou estratégias específicas de regularização e inicialização.

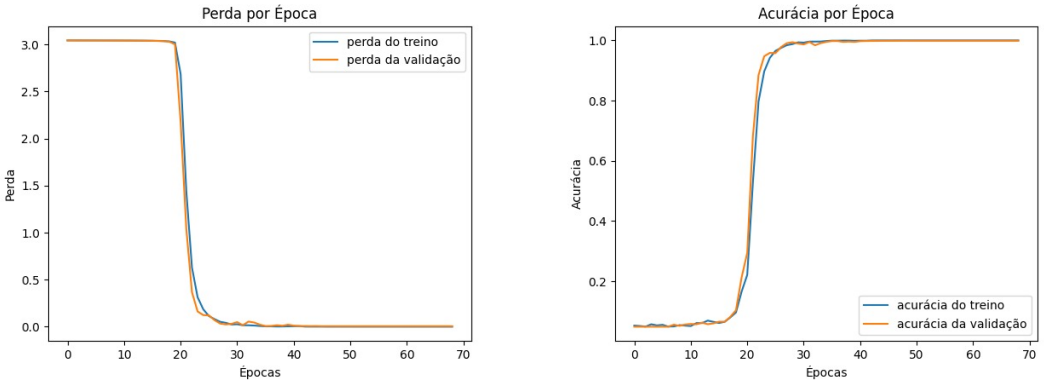
O modelo foi interrompido um pouco antes da amostra 70 devido à ativação da função de *early stopping*, que interrompe o treinamento quando a desempenho do conjunto de validação para de melhorar, prevenindo o *overfitting*. Porém, não é um problema, considerando que ambas curvas diminuem pregressivamente e estão próximas umas das outras, comprovando não haver *overfitting* ou *underfitting*. A mesma análise pode ser feita no gráfico de acurácia por época na Figura 19, sendo que a acurácia inicial fica contante em zero e depois da época 30 estabiliza em 1. Portanto, este modelo também apresentou um ótimo desempenho e seu treinamento.

Figura 18 – Matriz Confusão — VGG16.



Fonte: Próprio Autor (2024).

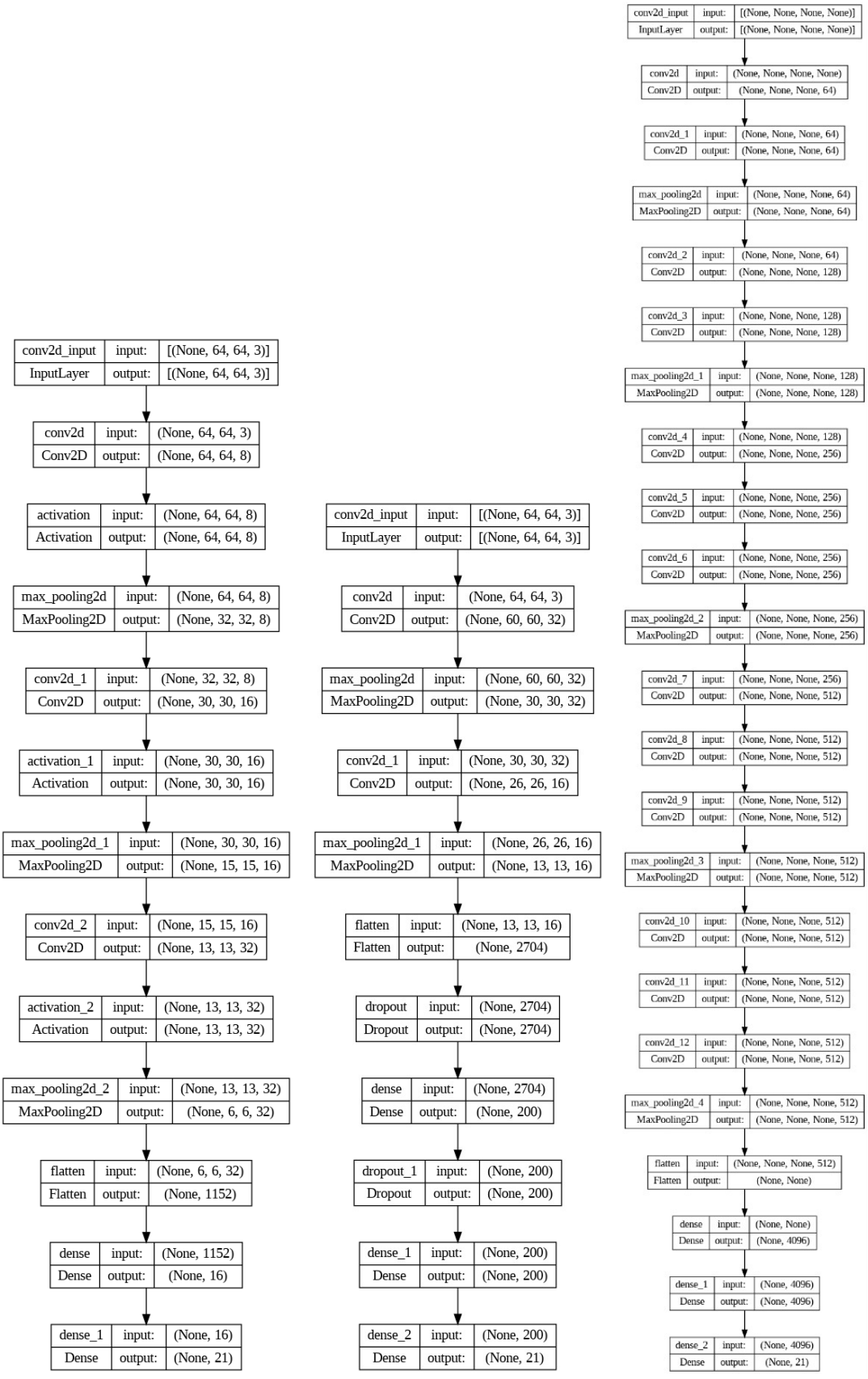
Figura 19 – Gráficos de comportamento do treinamento por época — VGG16.



(a) Perda por época. (b) Acurácia por época.

Fonte: Próprio Autor (2024).

Figura 20 – Arquiteturas das redes utilizadas.



(a) CNN. (b) FFCNN. (c) VGG16.

Fonte: Próprio Autor (2024).

Tabela 1 – Relatório de classificação do *dataset* de teste — CNN, FFCNN e VGG16.

	CNN			FFCNN			VGG16			
	PRECISION	RECALL	F1-SCORE	PRECISION	RECALL	F1-SCORE	PRECISION	RECALL	F1-SCORE	SUPPORT
A	0.99	0.99	0.99	1	1	1	1	0.99	1	454
B	0.99	1	0.99	1	1	1	1	1	1	446
C	0.99	0.99	0.99	1	1	1	1	1	1	455
D	0.98	0.98	0.98	1	0.99	1	1	1	1	440
E	0.99	0.98	0.99	1	1	1	1	0.99	1	450
F	0.99	0.99	0.99	1	1	1	1	1	1	421
G	0.99	0.98	0.99	0.99	1	0.99	0.99	1	1	440
I	0.98	1	0.99	1	1	1	1	1	1	440
L	0.99	1	1	1	1	1	1	0.99	1	440
M	0.99	1	0.99	1	0.99	1	1	1	1	440
N	1	0.99	0.99	0.99	1	0.99	1	1	1	440
O	0.99	0.98	0.99	1	1	1	0.99	1	0.99	440
P	1	1	1	1	1	1	1	1	1	440
Q	1	1	1	1	0.99	0.99	1	1	1	440
R	0.99	1	1	1	1	1	1	1	1	440
S	0.99	1	1	1	1	1	1	1	1	440
T	1	1	1	1	1	1	0.99	1	1	434
U	1	1	1	1	1	1	1	1	1	440
V	1	1	1	1	1	1	1	1	1	440
W	1	1	1	1	1	1	1	1	1	440
Y	1	1	1	1	1	1	1	1	1	440
Média	0.99	0.99	0.99	1	1	1	1	1	1	422.14
Total	NA	NA	NA	NA	NA	NA	NA	NA	NA	11548

Fonte: Próprio Autor (2024).

4.4 RECONHECIMENTO EM TEMPO REAL

O reconhecimento em tempo real integrado com sintetizador de voz foi testado com todas as letras, apresentando um bom resultado, considerando que os três modelos retornaram as predições corretas e a letra sintetizada corretamente. Porém, em sinais parecidos os gestos precisam ser bem feitos de modo que não fique parecido com outro sinal, além disso, a qualidade da *webcam*, a luz do ambiente e a distância da câmera são fatores que influenciam nas predições.

Ao fazer o sinal, conforme mostra a Figura 21, o algoritmo retornou a predição e sintetizou a voz de cada modelo. Em paralelo, o algoritmo salva as imagens com a predição, conforme mostra as imagens da Figura 22. Foram testadas todas as letras, porém, não se faz necessário apresentar todas as imagens.

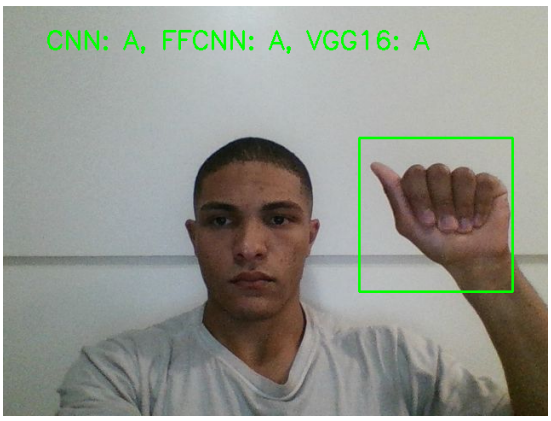
Figura 21 – Interface da aplicação de reconhecimento e tempo real.

Fonte: Próprio Autor (2024).

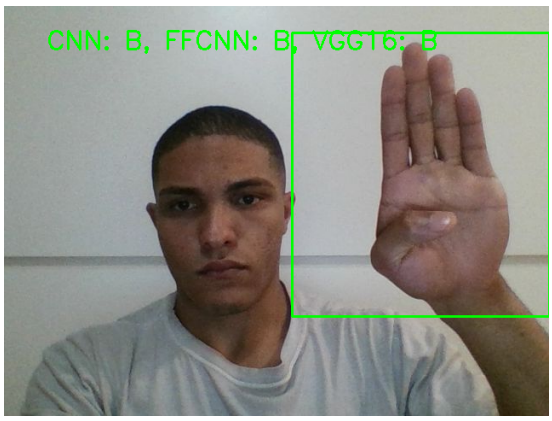
4.5 CONSIDERAÇÕES FINAIS

Os resultados obtidos no presente trabalho mostram um alinhamento consistente com os objetivos propostos, tanto geral quanto específicos. O objetivo geral de desenvolver um modelo computacional capaz de traduzir sinais da Libras para o português foi amplamente alcançado. Utilizando técnicas de redes neurais para a tarefa de classificação de 21 gestos do alfabeto manual em Libras, o modelo foi desenvolvido inteiramente em

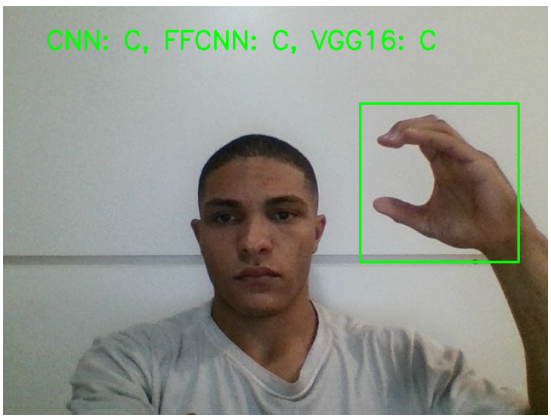
Figura 22 – Imagem de saída do algoritmo de reconhecimento em tempo real.



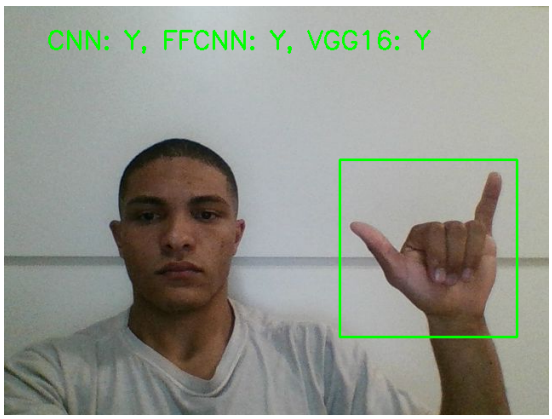
(a) Imagem de saída - letra A.



(b) Imagem de saída - letra B.



(c) Imagem de saída - letra C.



(d) Imagem de saída - letra Y.

Fonte: Próprio Autor (2024).

uma plataforma em nuvem, eliminando a necessidade de hardware local e comprovando sua viabilidade e eficiência.

Em relação aos objetivos específicos, o estudo bibliográfico de trabalhos relacionados e referencial teórico proporcionou uma base teórica sólida, identificando as melhores práticas, desafios e tecnologias existentes. A aplicação de técnicas de redes neurais para reconhecer padrões de Libras foi executada com sucesso, com os modelos treinados mostrando excelente desempenho nas métricas de avaliação utilizadas. O reconhecimento de Libras em tempo real, conforme mostrado acima, foi testado e validado utilizando uma webcam, demonstrando a viabilidade prática do sistema. Por fim, a implementação do sintetizador de voz, integrando a *Web Speech API*, permitiu converter os sinais de Libras reconhecidos em fala, oferecendo uma interface de usuário auditiva eficiente.

Com todos os objetivos alcançados e os resultados satisfatoriamente analisados, concluí-se a seção de resultados. Os métodos e técnicas aplicados mostraram-se eficazes para a tradução automática de Libras em tempo real, com integração de síntese de voz, validando a proposta inicial deste trabalho. Na próxima seção, será apresentada as conclusões finais, discutindo as implicações dos resultados obtidos e possíveis direções

para pesquisas futuras.

5 CONCLUSÃO

O presente trabalho de conclusão de curso abordou técnicas de aprendizado de máquina, desenvolvendo e aplicando redes neurais convolucionais (CNN), a arquitetura VGG16 e a Fast Filtered Convolutional Neural Network (FFCNN) para a tarefa de reconhecimento de gestos estáticos, especificamente, 21 letras do alfabeto de Libras. Além disso, para validar os modelos treinados, eles foram submetidos a um *dataset* de teste tendo suas respectivas métricas de avaliação analisadas. Por fim, foi realizada uma integração em ambiente real, utilizando uma webcam para avaliar o reconhecimento de gestos em tempo real e um sintetizador de voz.

Ao submeter o *dataset* de teste, os modelos apresentaram resultados próximos e satisfatórios. A rede CNN apresentou uma acurácia de 99,3%, e em algumas classes as métricas de classificação foi menor que as demais. Já a FFCNN obteve uma acurácia de 99,9% e apresentou excelentes métricas de classificação, atingindo 100% em praticamente todas as classes. Por fim, a VGG16 retornou 99,8% de acuracidade e suas classificações foram excelentes, bem similar ao FFCNN. Portanto, a rede FFCNN obteve a maior acurácia dentre os modelos e o maior tempo de treinamento, porém, dado a pequena diferença de acurácia entre os modelos, as três soluções são viáveis para reconhecimento de sinais estáticos.

Comparando os resultados deste trabalho com os correlatos, (SILVA, K. P. d., 2023) apresentou 100% de acurácia com redes LSTM e Mediapipe, e (SILVA, R. P. d. *et al.*, 2022) obteve êxito ao utilizar MediaPipe, embora em um conjunto limitado de sinais. Além disso, (CRUZ, 2019) apresentou 99,45% de acurácia para a rede FFCNN. Portanto, pode-se afirmar que o presente trabalho entregou métricas comparáveis as dos correlatos para as três redes analisadas, além de possuir um diferencial ao integrar um sintetizador de voz para tradução em tempo real e ser implementado em uma plataforma em nuvem.

O desenvolvimento em uma plataforma em nuvem (*Google Colab*) foi algo inovador para este tipo de trabalho e de extrema importância, considerando a indisponibilidade de uma máquina com bom desempenho para treinar os modelos. A plataforma mostrou-se versátil para aplicações de Aprendizado de Máquina e Inteligência Artificial.

Durante o desenvolvimento da presente pesquisa, foram enfrentados diversos desafios. Um dos principais foi treinar os modelos até atingir valores de acurácia aceitáveis, já que o processo de treinamento é demorado e não existe um método para determinar os valores ideais dos parâmetros. A limitação de tempo de uso gratuito da GPU no *Google Colab* também contribuiu para aumentar o tempo necessário para o treinamento das redes neurais. Implementar a leitura em tempo real da *webcam* no *Colab* se mostrou complexo, uma vez que as funções do *OpenCV* (cv2) usadas em visão computacional executam com algumas diferenças e limitações nessa plataforma, exigindo a busca e adaptação através biblioteca ou aplicações em *Javascript* que auxiliam na captura dos *frames* em tempo

real. Além disso, o reconhecimento em tempo real apresentou erros inicias nas predições, tornando necessário realizar alguns ajustes, como usar fundo branco nos *frames* com o sinal, recortar a mão no tamanho correto utilizando os pontos centrais da mão, espelhar horizontalmente a imagem, ajustar a posição e a claridade da webcam, normalização da imagem, ajustes de dimensão entre outros ajustes no pré-processamento. Portanto, esses desafios foram superados com suporte do professor orientador, fóruns, livros técnicos, análise de projetos similares e iterativas melhorias.

Como propostas de trabalhos futuros, seria utilizar outros tipos de redes neurais, como a LSTM com o intuito de predizer sinais dinâmicos, os quais precisam de um conjunto de *frames* para o reconhecimento de apenas um sinal, podendo assim abranger um vocabulário maior de libras. Adicionalmente, implementar outros tipos de arquitetura como as Redes Recorrentes (RNNs) e suas variantes (LSTM, GRU), eficazes para modelar dependências temporais em sequências de dados; a combinação de Redes Neurais Convolucionais 2D com LSTM (ConvLSTM), que une benefícios espaciais e temporais; as Redes Neurais Adversariais Generativas (GANs), úteis para gerar sequências realistas de gestos de sinais de libras. Além das arquiteturas, é possível explorar outros frameworks e modelos pré-treinados para o reconhecimento de sinais de libras em vídeos. Exemplos incluem o *YOLO* para detecção em tempo real de objetos, o RCNN para segmentação de objetos, o *OpenPose* para detecção de poses humanas e o *DeepLab* para segmentação semântica precisa. Esses frameworks oferecem uma gama de abordagens complementares que podem ser utilizadas em trabalhos futuros para melhorar o reconhecimento de sinais de libras em vídeos.

REFERÊNCIAS

ALBUQUERQUE, Márcio P de; CANER, ES; MELLO, AG; ALBUQUERQUE, MP. Análise de imagens e visão computacional. **Centro Brasileiro de Pesquisas Físicas. Rio de Janeiro**, 2012.

BALLARD, Dana Harry; BROWN, Christopher M. **Computer vision**. [*S.l.*]: Prentice Hall Professional Technical Reference, 1982.

BARELLI, Felipe. **Introdução à Visão Computacional: Uma abordagem prática com Python e OpenCV**. [*S.l.*]: Editora Casa do Código, 2018.

BODEN, Margaret A. **Artificial intelligence**. [*S.l.*]: Elsevier, 1996.

BRAGA, A. **Redes neurais artificiais: teoria e aplicações.** [*S.l.*]: [S.l.]: LTC editora Rio de Janeiro, Brazil, 2007.

CHEVTCHENKO, Sérgio F; VALE, Rafaella F; MACARIO, Valmir; CORDEIRO, Filipe R. A convolutional neural network with feature fusion for real-time hand posture recognition. **Applied Soft Computing**, Elsevier, v. 73, p. 748–766, 2018.

CONCI, Aura; AZEVEDO, Eduardo; LETA, Fabiana R. Computaçao gráfica–teoria e prática,[v. 2]. **Rio de Janeiro**, 2008.

CRUZ, Lisandra Sousa da. **Comparação de algoritmos de reconhecimento de gestos aplicados à sinais estáticos de Libras**. 2019. B.S. thesis – Brasil.

DUTOIT, Thierry. **An introduction to text-to-speech synthesis**. [*S.l.*]: Springer Science & Business Media, 1997. v. 3.

FLECK, Leandro; TAVARES, Maria Hermínia Ferreira; EYNG, Eduardo; HELMANN, Andrieli Cristina; ANDRADE, MA de M. Redes neurais artificiais: Princípios básicos. **Revista Eletrônica Científica Inovação e Tecnologia**, v. 1, n. 13, p. 47–57, 2016.

HONORA, Márcia. **Livro Ilustrativo da Língua Brasileira de Sinais vol. 2: Desvendando a comunicação usada pelas pessoas com surdez**. [*S.l.*]: Ciranda Cultural, 2020. v. 2.

KAMARAINEN, J-K; KYRKI, Ville; KALVIAINEN, Heikki. Invariance properties of Gabor filter-based features-overview and applications. **IEEE Transactions on image processing**, IEEE, v. 15, n. 5, p. 1088–1099, 2006.

LACERDA, Lucas. **Redes Neurais Convolucionais - LIBRAS**. 2019. Disponível em: https://github.com/lucaaslb/cnn-libras.

LECUN, Yann; KAVUKCUOGLU, Koray; FARABET. Convolutional networks and applications in vision. *In*: IEEE. PROCEEDINGS of 2010 IEEE international symposium on circuits and systems. [*S.l.*: *s.n.*], 2010.

LUCAS. **Utilizando a arquitetura VGG para a classificação de doenças nas plantas de feijão**. 2021. Disponível em: https://lucas208.medium.com/utilizando-a-arquitetura-vgg-para-a-classifica%C3%A7%C3%A3o-de-doen%C3%A7as-nas-plantas-de-feij%C3%A3o-75c5ae6cea88.

MARQUES, Willian. **Data Set Libras**. 2022. Disponível em: https://github.com/WillJR183/analyse-classifiers-libras/tree/master/dataset_full.

MEDIUM. **Utilizando a arquitetura VGG para a classificação de doenças nas plantas de feijão**. 2021. Disponível em: https://lucas208.medium.com/utilizando-a-arquitetura-vgg-para-a-classifica%C3%A7%C3%A3o-de-doen%C3%A7as-nas-plantas-de-feij%C3%A3o-75c5ae6cea88.

MITCHELL, Tom M. Does machine learning really work? **AI magazine**, v. 18, n. 3, p. 11–11, 1997.

NIELSEN, Michael. **Neural Networks and Deep Learning Book**. 2019. Disponível em: http://neuralnetworksanddeeplearning.com/chap6.html.

PEDRINI, Hélio; SCHWARTZ, William Robson. **Análise de imagens digitais: princípios, algoritmos e aplicações**. [*S.l.*]: Cengage Learning, 2008.

RESEARCHGATE. **Development Control Parking Access Using Techniques Digital Image Processing And Applied Computational Intelligence**. 2015. Disponível em: https://www.researchgate.net/publication/271841344_13TLA1_39CavalcantiNeto.

SANTOS, Calebe Augusto dos; DORNELLES, Gino. Text-to-speech: Sintetizador de voz para documentos como ferramenta de apoio a deficientes visuais, 2008.

SANTOS ANJO, Mauro dos; PIZZOLATO, Ednaldo Brigante; FEUERSTACK, Sebastian. A real-time system to recognize static gestures of Brazilian sign language (libras) alphabet using Kinect. *In*: IHC. [*S.l.*: *s.n.*], 2012. P. 259–268.

SILVA, Karolyne Pereira da. ANÁLISE DE APLICAÇÃO DE VISÃO COMPUTACIONAL E REDES NEURAIS, EM CONJUNTO COM O USO DE TÉCNICAS DE AUMENTO DE DADOS, NA TRADUÇÃO AUTOMÁTICA DE LIBRAS, 2023.

SILVA, Romário Pereira da *et al.* Visão computacional: um estudo de caso aplicado a língua brasileira de sinais (LIBRAS). Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, 2022.