Fabio Augusto Ramalho, Rafael Silva Mamede, Victor Daisuke Dano

DeepUAI - Interface gráfica para aplicações de Redes Neurais

Brasil

Dezembro de 2022

Fabio Augusto Ramalho, Rafael Silva Mamede, Victor Daisuke Dano

DeepUAI - Interface gráfica para aplicações de Redes Neurais

Monografia referente ao Trabalho Final de Graduação do curso de Engenharia de Computação da Universidade Federal de Itajubá

Universidade Federal de Itajubá – UNIFEI

Instituto de Engenharia de Sistemas e Tecnologia da Informação

Engenharia de Computação

Orientador: Edvard Martins De Oliveira

Brasil

Dezembro de 2022

Dedicamos este trabalho primeiramente às nossas famílias,
que nos apoiaram desde o início do curso.

Dedicamos também ao corpo docente do curso de Engenharia da Computação da UNIFEI
Itajubá, eles nos deram toda a base necessária para o desenvolvimento deste trabalho.

Agradecimentos

Agradecimento especial ao orientador deste trabalho, Prof. Edvard Martins de Oliveira, pelo auxílio na sintetização da ideia inicial, orientação quanto a estrutura da monografia e correções dos textos. Sem dúvida os conselhos foram de enorme ajuda para o desenvolvimento deste trabalho.

Resumo

A Inteligência Artificial (IA) e os algoritmos de *machine learning* foram criados e desenvolvidos ao longo de anos para trazer benefícios e avanços importantes para a sociedade. Fazer o uso de *machine learning* não é uma tarefa simples, diversas ferramentas atuais demandam um conhecimento prévio em programação e acabam sendo muito restritas. Este trabalho tem como objetivo criar uma interface gráfica *web* para que usuários possam experimentar o funcionamento de um processo de *deep learning* de forma interativa. O sistema foi desenvolvido por meio do uso de bibliotecas na linguagem *Python* e uma ferramenta de criação de interfaces gráficas na web, o *ReactJS*. O usuário poderá interagir com a interface sem a necessidade de escrever códigos. O sistema DeepUAI pode ser utilizado para classificar, treinar e visualizar resultados com a escolha do *dataset* e, posteriormente, o envio da imagem que o usuário deseja classificar. O primeiro experimento realizado foi por meio do uso de um *dataset* de classificação de frutas, após o treinamento da máquina, ela foi capaz de classificar um abacate com sucesso. A linguagem de sinais americana (ASL - American Sign Language) também foi um objeto de estudo, nele é esperado que a aplicação, dada a imagem, identifique a letra corretamente. A aplicação conseguiu acertar cerca de 66,67% das letras testadas. Era esperado um resultado mais preciso, porém diversos fatores referentes ao treinamento com o conjunto de dados influenciaram no processo de *deep learning*.

Palavras-chave: interface gráfica. *dataset*. *machine learning*. *deep learning*. inteligência artificial. classificação de imagens.

Abstract

Artificial Intelligence (AI) and machine learning algorithms were created and developed through the years to bring benefits and important advances to society. Using machine learning is not a simple task, several current tools demand previous programming knowledge and thus becoming very restricted. The objective of this work is to create a web graphical user interface so users can experience the operation of a deep learning process in an interactive way. The system was developed using *Python* language libraries and a web graphical user interface creation tool, the *ReactJS*. The user will be able to interact with the interface without coding. The DeepUAI system can be used to classify, train and visualize the results by choosing the dataset for training and lately sending the image that the user wants to classify. The first experiment was performed using a fruit classification dataset. After the training process, the machine was able to successfully classify the avocado. American Sign Language (ASL) was also an object of study, it is expected that the application, given the letter image, identify the letter correctly. The application managed to hit about 66.67% of the letters tested, a more accurate result was expected, but several factors related to training with the dataset influenced deep learning process.

Keywords: graphical user interface. dataset. machine learning. deep learning. artificial intelligence. image classification.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
ASL	American Sign Language
CNN	Convolutional Neural Network
DBN	Deep Belief Network
deconvnet	Deconvolutional Network
DNN	Deep Neural Networks
DOM	Document Object Model
DSVC	Data Version Control System
DVC	Data Version Control
GAD	Grafo Acíclico Dirigido
GPU	Graphics Processing Units
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ML	Machine Learning
MLdp	Machine Learning data platform
MVC	Model View Controller
NPM	Node Package Manager
POO	Programação Orientada à Objetos
RNN	Recurrent Neural Network
SciANN	Scientific Computing with Artificial Neural Networks
SPA	Single Page Application

Sumário

1 Introdução

O avanço da computação ao longo de décadas apresentou muitas tecnologias. Uma delas é a inteligência artificial, área que engloba muitas ferramentas que já são amplamente utilizadas por empresas e cientistas, e vêm ganhando cada vez mais espaço. IA é um termo muito amplo, pois existem inúmeras aplicações que usam e evoluem o conceito desta área.

Segundo Rouhiainen (2018), a IA é definida como a habilidade dos computadores para realizar atividades que normalmente requerem inteligência humana. De forma mais detalhada, é a capacidade das máquinas de executarem algoritmos e aprender com base nos dados e tomar decisões através do aprendizado. As tecnologias baseadas em IA vêm sendo utilizadas para ajudar seres humanos em diversos âmbitos da vida, mas é preciso se atentar também aos possíveis impactos negativos dessa rápida proliferação.

No fim da Segunda Guerra Mundial já haviam registros sobre mecanismos que imitavam ações humanas, além do avanço no estudo sobre o cérebro humano dirigido por médicos e psicólogos. No ano de 1948, houve um encontro conhecido como Simpósio de Hixon, que reuniu pesquisadores de diversas áreas com o objetivo de tentar compor o funcionamento da mente humana. Em 1950, Alan Turing desenvolveu uma máquina capaz de emular a comunicação escrita de um humano e publicou o artigo *Computing Machinery and Intelligence*, considerado o texto que fundou a IA (BARBOSA, 2020). Os anos seguintes se sucederam com novas invenções até que o termo *Machine Learning* (ML) foi utilizado pela primeira vez como um sistema que permite computadores aprender uma função, sem serem programados diretamente para tal finalidade.

Uma das áreas que explora o uso de IA é a área de *marketing*, com o crescente volume de dados gerados atualmente, de modo que, em diversas situações, seria inviável retirar informações úteis sem o uso de *machine learning* em tempo real. Algumas aplicações populares de IA em *marketing* são: criação de conteúdo, pesquisa por voz, análise preditiva, personalização de anúncios, *chatbot*, entre outros. Por exemplo, o *chatbot* é considerado um *software* de inteligência artificial que pode ajudar a simular conversas de *chats* com usuários. O *chatbot* pode identificar rapidamente as intenções do usuário com base nas palavras recebidas, reduzindo o tempo e custo de serviços ao consumidor, um grande benefício do uso desta tecnologia (MITIĆ et al., 2019).

Outra aplicação muito comum de IA está na classificação de imagens por meio de máquinas treinadas previamente por conjunto de dados, um dos focos de desenvolvimento deste trabalho. Por exemplo, o sistema Google Lens[1], anunciado pela Google em 2017, permite a identificação de objetos apenas apontando a câmera. Ele é capaz de mostrar o

[1] Disponível em *smartphones* e <lens.google/intl/pt-BR/>

nome de plantas e animais, tradução de escritas em outro idioma e até realizar a pesquisa do objeto no *Google Shopping*, a ferramenta de consulta de preços da própria empresa. A classificação de imagens pode até salvar vidas: no trabalho realizado por Wurzel e Martins (2022), o *machine learning* foi utilizado para identificar câncer de mama em imagens de mamografia, classificando o potencial de casos positivos e apresentando uma acurácia de 91,34%. A detecção precoce é importante para garantir que a mulher consiga passar pelo tratamento e combater o câncer com sucesso. E estes são apenas exemplos de como a IA pode ajudar a sociedade.

1.1 Apresentação

No meio acadêmico existem diversas pesquisas que citam a democratização da inteligência artificial. Finley (2019) escreveu sobre um projeto da Google para democratizar a IA por meio de *kits* contendo um Rasberry Pi e pequenos alto falantes que, em conjunto, imitam sistemas como Amazon Echo e Google Home: dispositivos que contém assistentes inteligentes para realizar comandos e ações. Com o uso da linguagem Python é possível personalizar o aparelho deste *kit*, gerando cada vez mais interesse ao recém chegado neste mundo de inteligência artificial.

Uma outra abordagem foi realizada por Sikpa et al. (2019) em sua pesquisa sobre a detecção e quantificação de metástases cerebrais por câncer de mama em animais, utilizando ferramentas democratizadas de *machine learning*. A ferramenta utilizada foi o TWS, *Trainable Weka Segmentation*, uma ferramenta de código aberto para aprendizado de máquina e mineração de dados da plataforma Fiji. Com a utilização correta do TWS, em conjunto com o conhecimento matemático dos autores, o classificador foi treinado até obter resultados com altos índices de acerto. Os autores destacam que a aplicação pode ser usada por não especialistas, com tutoriais e dados disponíveis para serem visualizados e reproduzidos.

Com base nos trabalhos anteriores, foi possível notar o esforço de pesquisadores para democratizar e facilitar o uso de inteligência artificial. A ideia inicial deste projeto surgiu com a pergunta "Seria possível criar uma forma das pessoas sem conhecimento em programação utilizarem *machine learning*?", visto que utilizar sistemas que envolvem inteligência artificial nem sempre é simples. Desta forma, neste trabalho é proposto um sistema para democratizar e facilitar o uso de *machine learning* para pessoas sem conhecimento de programação, por meio de uma interface gráfica amigável ao usuário que permitirá realizar classificações e treinamentos de máquina.

1.2 Justificativa

Segundo um estudo de mercado realizado pela Furtunate Business Insights (2022, *tradução livre*), espera-se que o mercado de *machine learning* cresça de US$ 21,17 bilhões em 2022 para US$ 209,91 bilhões em 2029, com taxa de crescimento anual composta de 38,8%. Baseado neste mesmo estudo, o mercado de *machine learning* exibiu um crescimento de 36,1% no ano de 2020, em comparação com o de 2019. O autor também argumenta que a pandemia de *COVID-19* teve grande impacto no aumento da demanda por essa tecnologia.

Com o crescimento do mercado de *machine learning*, é esperado que o número de pessoas interessadas em desenvolver suas próprias aplicações de redes neurais também aumente. Tal grupo de pessoas inclui desde estudantes e profissionais da área de tecnologia da informação, até entusiastas e leigos que buscam uma solução simples para seus problemas do cotidiano e de seus negócios.

Para contemplar a variedade de nível de conhecimento técnico do público que entrará em contato com o *machine learning* na próxima década, será de extrema importância o surgimento de novas ferramentas para criação dessas aplicações, contemplando desde a possibilidade de criação de algoritmos cada vez mais complexos e precisos, até a evolução das ferramentas de construção, para que se tornem cada vez mais simples de serem utilizadas.

Há diversas vantagens em se utilizar interfaces gráficas para o uso de ferramentas digitais. No texto de CHTips (2022, *tradução livre*), o autor levanta uma lista de vantagens do uso de interfaces gráficas em aplicações para computadores, dentre elas:

- São fáceis de usar e simples de entender, portanto substituem sistemas antiquados que utilizam interfaces de comandos;
- Não requerem habilidades ou conhecimentos em computação para realizar operações;
- Como os sistemas operacionais modernos são principalmente utilizados através de interfaces gráficas, a comunicação com o usuário pode ser estabelecida mais facilmente.

1.3 Objetivo

Esta seção apresenta o objetivo geral e os específicos da pesquisa.

1.3.1 Objetivo Geral

Construir uma interface *web* para facilitar e agilizar o uso de redes neurais para classificação de dados; e a criação de novas aplicações de redes neurais através do treinamento de modelos sob os dados disponibilizados pelos usuários.

1.3.2 Objetivos Específicos

- Permitir ao usuário, por meio da interface, escolher entre a utilização de *datasets* previamente disponibilizados ou importados por ele para realizar o treinamento das redes neurais.
- Disponibilizar modelos de redes neurais prontos para serem treinados e utilizados para classificação.
- Possibilitar a classificação de qualquer imagem enviada pelo usuário por diferentes aplicações de redes neurais.

1.4 Estrutura da Monografia

A monografia está estruturada em cinco seções: Introdução, Revisão Bibliográfica, Desenvolvimento, Resultados e Conclusão.

Nesta primeira seção foi feita uma apresentação sobre o que é a inteligência artificial, sua abrangência, importância e evolução ao longo dos anos. Na introdução ainda foram apresentadas diversas aplicações que representam um grande avanço na tecnologia e na forma de pensar sobre o uso de *machine learning*. Além disso, foram apresentados os objetivos, justificativa e discussão do problema.

O Capítulo 2, de referência bibliográfica, traz um estudo do estado da arte na área de IA, *machine learning*, versionamento e modelos de redes neurais. No final da seção são apresentados diversos trabalhos relacionados, destacando as semelhanças e diferenças destes trabalhos com o DeepUAI.

No Capítulo 3, chamado de desenvolvimento, é demonstrada a metodologia utilizada para a construção do sistema, explicando a escolha das ferramentas e bibliotecas para a implementação da interface gráfica e do *back-end* da aplicação. Nesta seção, ainda se encontram diagramas que demonstram casos de uso, fluxo de funcionalidades, arquitetura do sistema e entidade e relacionamento. Essas figuras auxiliam na compreensão da estrutura do sistema.

No Capítulo 4, referente à resultados, são apresentadas capturas de tela da interface gráfica do sistema DeepUAI, demonstrando detalhadamente as ações que o usuário pode

performar. Estão presentes os *datasets*, os modelos de redes neurais e as aplicações, que consistem no treinamento das redes neurais com a utilização dos conjuntos de dados.

Por fim, é feita uma recapitulação das contribuições do trabalho e comentários sobre os objetivos e resultados atingidos. A conclusão ainda traz um trecho dedicado a trabalhos futuros, que podem ser explorados por outros pesquisadores e enriquecer ainda mais os estudos na área de *machine learning*.

2 Revisão Bibliográfica

2.1 Inteligência Artificial, Machine Learning e Deep Learning

A IA, de forma simplista, representa o estudo da construção de máquinas com sentidos humanos, que analisam, entendem e respondem. A popularidade dos últimos anos se deu por três motivos: primeiro, a disponibilidade de *big data*, uma quantidade enorme de dados gerados em diversas áreas; em seguida os algoritmos de *machine learning*, que se aprimoraram e tornaram-se mais confiáveis; e por fim, o uso de computação em nuvem e o avanço em performance dos sistemas computacionais trouxe a possibilidade de execução de algoritmos mais complexos em tempo hábil (MONDAL, 2020).

Machine learning é uma aplicação de inteligência artificial que permite sistemas aprenderem e se desenvolverem através de experiências, sem estar sendo programado (THE ROYAL SOCIETY, 2017). É utilizado em uma grande variedade de aplicações, de reconhecimento de fala até assistentes inteligentes, carros autônomos, cuidados com a saúde, entre outros (MARQUES; WANGENHEIM; HAUCK, 2020).

Deep learning permite que modelos computacionais compostos por múltiplas camadas de processamento aprendam as representações dos dados com múltiplos níveis de abstração. Permitiu avanços em diversas áreas, como: reconhecimento de fala e de objetos; e até na saúde, com a descoberta de novos medicamentos. O *deep learning* descobre estruturas complexas em uma grande base de dados, utilizando algoritmos de realimentação (*backpropagation*) para indicar como a máquina deve modificar os parâmetros utilizados na representação de cada camada em relação à anterior (LECUN; BENGIO; HINTON, 2015).

2.2 Arquitetura de Redes Neurais Convolucionais

Tendo em vista a frequente aplicação dos conceitos de inteligência artificial para tarefas de reconhecimento e classificação de imagens, uma das três maiores arquiteturas de *deep learning* vêm se destacando, as Redes Neurais Convolucionais. Basicamente, dentro desse importante campo de IA, que são as redes neurais profundas, tem-se as *Deep Belief Networks* (DBN), empregadas principalmente no processamento de linguagem natural; as *Recurrent Neural Networks* (RNN), comumente aplicadas no processamento de linguagem, texto e fala; e as *Convolutional Neural Networks* (CNN), destinadas às tarefas de classificação e reconhecimento em imagens (PATEL, 2020).

Dentro da CNN, diversas formas de implementação vêm surgindo, com o objetivo de melhorar a performance das redes para a execução de sua determinada função inteligente.

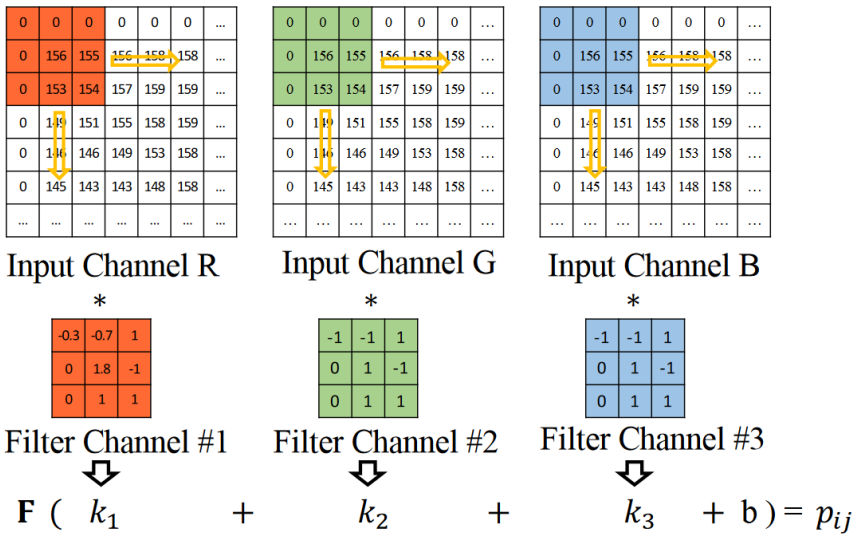
Patel (2020) cita os modelos LeNet, AlexNet, ZFNet, GoogleNet, VGGNet e ResNet em sua pesquisa, os quais serão abordados na próxima seção. Além destes, outras arquiteturas podem ser encontradas na literatura, como é o caso dos padrões EfficientNet e Xception.

De modo simplificado, todos os diferentes modelos de CNN são desenvolvidos a partir do empilhamento de determinados elementos e camadas, como *convolutional layers*, *non-linearity layers*, *pooling layers* e *fully-connected layers* (ALBAWI; MOHAMMED; ALZAWI, 2017).

2.2.1 Convolutional Layers

As camadas convolucionais são aquelas responsáveis por aplicar determinado filtro, também conhecido por kernel, nos dados de sua entrada, a fim de transformar as informações buscando identificar determinada característica. Assim, o filtro é aplicado de forma semelhante a uma janela, percorrendo toda a matriz de dados e mapeando-os para os conhecidos *feature maps* (Figura 1).

Figura 1 – Exemplo de convolução.

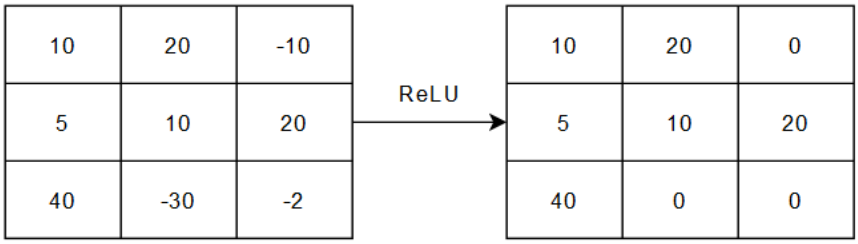


Fonte: Qin et al. (2018).

2.2.2 Non-Linearity Layers

As camadas não lineares normalmente vêm após as camadas convolucionais na arquitetura da rede neural e possuem o objetivo de aplicar determinada função nas informações, de forma a melhor ajustá-las para a saída desejada. Um exemplo de equação matemática amplamente utilizada para esses casos é a função ReLU, que basicamente zera o valor dos elementos negativos e mantém os demais (Figura 2).

Figura 2 – Exemplo de aplicação da função ReLU.

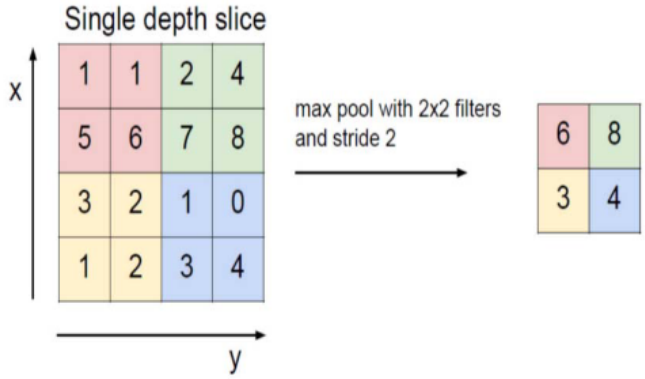


Fonte: Autoria própria, 2022.

2.2.3 Pooling Layers

A ideia das camadas de *pooling*, em resumo, é diminuir a dimensão dos dados, reduzindo assim sua complexidade e facilitando o trabalho para as camadas posteriores. Para isso, existem alguns tipos de operações que podem ser tomadas, como por exemplo *max-pooling* e *average-pooling*. No caso do *max-pooling*, o redimensionamento ocorre através da captura dos maiores elementos, tendo em vista uma janela a ser percorrida na matriz de dados (Figura 3), enquanto que, para o *average-pooling*, uma média dos elementos da janela é realizada e os valores são capturados.

Figura 3 – Demonstração da operação *max-pooling*.



Fonte: Albawi, Mohammed e ALZAWI (2017).

2.2.4 Fully-Connected Layers

Já as *fully-connected layers* são aquelas camadas nas quais todos os perceptrons são conectados a todos os elementos da camada anterior e posterior. De modo geral, elas são responsáveis pela fase final de classificação e, o número de perceptrons da última camada, também é número de rótulos que determinado modelo consegue classificar.

2.3 Modelos de CNN

Considerando a presença, dentro da literatura, de vários modelos de CNN com diferentes arquiteturas, torna-se importante a tarefa de conhecer os padrões, suas principais aplicações e taxas de acerto, de modo a possibilitar com que o operador escolha aquele que melhor atenda sua necessidade. Outro ponto importante a ser analisado durante a escolha é onde determinado modelo de rede neural estará sendo empregado, uma vez que, caso seja em aplicações embarcadas, por exemplo, fatores como memória, poder de processamento e consumo de energia podem fazer a diferença na opção do modelo desejado.

2.3.1 LeNet

Lecun et al. (1998) propôs a arquitetura LeNet, também conhecida por LetNet-5. O modelo é empregado em tarefas de classificação de dígitos escritos a mão e é construído em cima de sete camadas, se não contabilizarmos a camada de entrada.

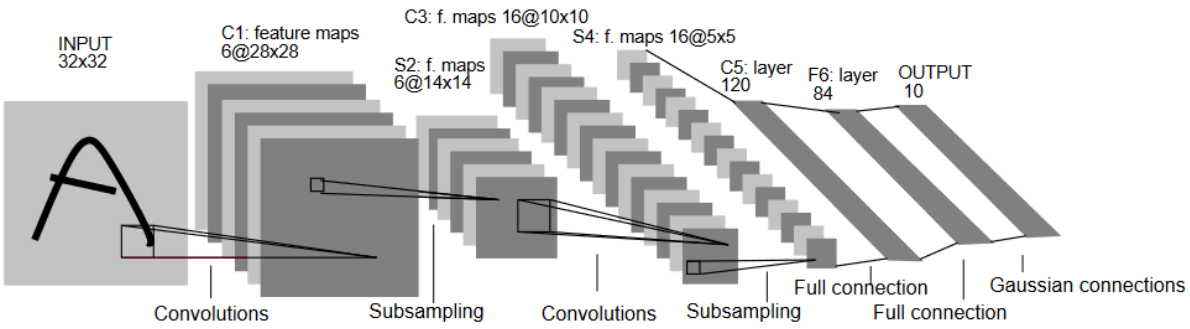
Esse padrão espera receber uma imagem de 32x32 *pixels*, onde a primeira camada é uma *convolutional layer* com seis *feature maps* de 28x28 *pixels*, os quais, de forma simplificada, são funções que possuem a responsabilidade de mapear os dados de sua entrada para outro espaço, a fim de facilitar a tarefa de classificação das informações. Assim, cada célula das 6x28x28 *feature maps* são conectadas a um quadro 5x5 da imagem de entrada.

A segunda camada é uma *pooling layer*, e especificadamente, uma *sub-sampling layer* com seis *feature maps* de 14x14 em que, cada unidade é ligada a uma janela 2x2 da primeira camada. Portanto, a função desta *layer* é reduzir o tamanho dos dados. Já a terceira é uma *convolutional layer* com 16 *feature maps*, seguida pela quarta camada, também responsável por realizar a subamostragem, como observado no padrão das duas primeiras. A quinta é uma *convolutional layer* com 120 *feature maps*, seguida por uma *fully-connected layer* de 84 unidades.

Essa descrição pode ser observada na Figura 4, onde as camadas convolucionais são representadas por Cx, as camadas responsáveis pela subamostragem por Sx e as camadas totalmente conectadas por Fx, além do elemento x variar conforme o índice da camada.

A taxa de erro referente aos testes da rede LeNet, utilizando 60.000 imagens do *dataset* MNIST, foi de 0,95%. Além de atingir uma taxa de erro de 0,8% quando acrescentou-se 540.000 novas imagens ao treinamento de forma artificial, realizando algumas manipulações nas imagens originais (LECUN et al., 1998).

Figura 4 – Arquitetura do modelo LeNet-5.



Fonte: Lecun et al. (1998).

2.3.2 AlexNet

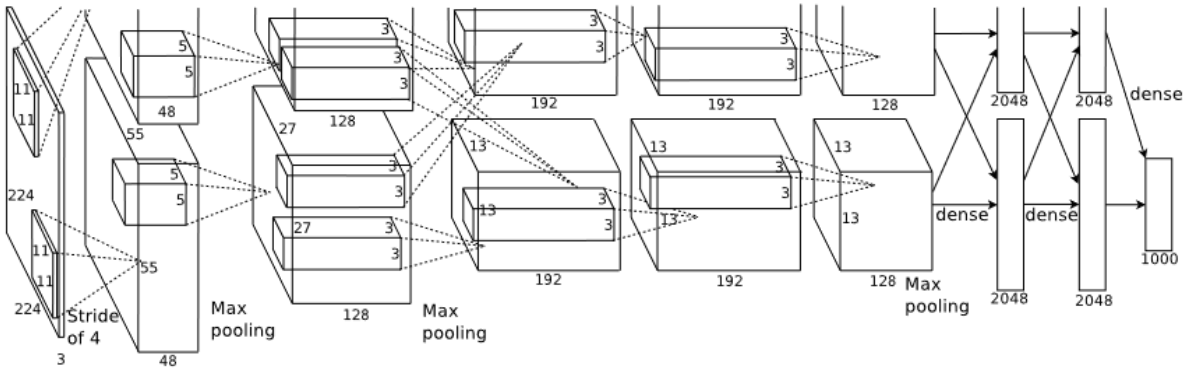
O modelo de rede neural convolucional proposto por Krizhevsky, Sutskever e Hinton (2012) surgiu em decorrência da recente disponibilidade de grandes *datasets* com imagens rotuladas e de alta resolução, como é o caso do ImageNet. Assim, para que seja possível esse aprendizado através de um modelo de CNN, tornou-se necessário o estudo de uma nova arquitetura.

Essa estrutura, composta por oito camadas, sendo as cinco primeiras do tipo *convolutional layer* e as próximas três *fully-connected layer*, pode ser observada na Figura 5. Além disso, uma característica importante desta rede é o fato dela ter sido desenvolvida para executar de forma paralela em duas diferentes *Graphics Processing Unit* (GPU), o que pode ser observado através da separação da arquitetura em uma parte superior e inferior na Figura 5. O resultado atingido nos testes, por esse padrão, tendo em vista a utilização do *dataset* ImageNet 2012, foi de taxas de erro top-1 igual a 37,5% e top-5 igual 17,0%, visualmente melhores do que o até então estado da arte, o qual possuía taxas de erro top-1 igual a 47,1% e top-5 igual a 28,2% (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Essas numerações top-1 e top-5 são relacionadas a comparação realizada a fim de chegar na determinada taxa de erro, sendo que a top-1 compara a verdadeira classificação da imagem com a primeira predição pelo modelo, enquanto que a top-5 relaciona o real rótulo com as cinco primeiras previsões. A fim de ranqueamento, a taxa top-5 normalmente é mais utilizada durante as competições do *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC).

2.3.3 ZFNet

O modelo ZFNet foi apresentado por Zeiler e Fergus (2013) através de um estudo que, inicialmente, buscou entender melhor, através da introdução de uma técnica de visualização que utiliza-se de *Deconvolutional Network* (deconvnet), o porque de determinada arquitetura atingir melhores taxas de acerto em comparação a outras. A partir disso, o padrão proposto foi de certa forma uma evolução das ideias introduzidas por Krizhevsky,

Figura 5 – Arquitetura do modelo AlexNet.

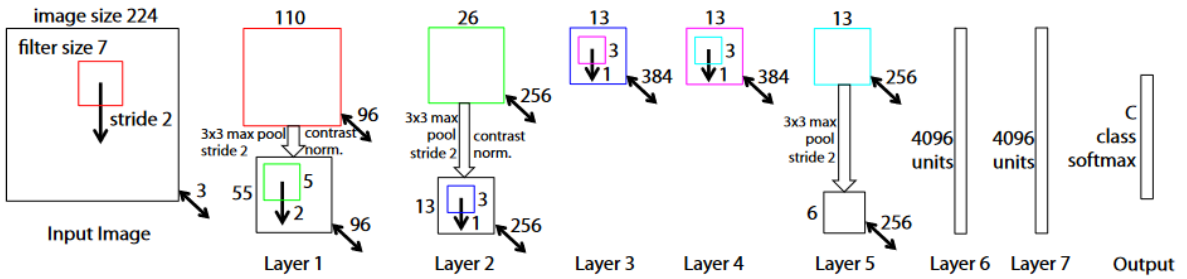


Fonte: Krizhevsky, Sutskever e Hinton (2012).

Sutskever e Hinton (2012) para o modelo AlexNet.

A composição do ZFNet foi pensada de modo a possuir oito camadas, como nota-se na Figura 6. Após a realização do treinamento da rede pelos autores do artigo, utilizando como entrada o *dataset* ImageNet 2012, foi alcançado uma taxa de erro top-5 de 14,8% relacionada aos testes.

Figura 6 – Arquitetura do modelo ZFNet.



Fonte: Zeiler e Fergus (2013).

2.3.4 VGGNet

Para a construção do modelo VGGNet, Simonyan e Zisserman (2014) basearam-se em um ponto específico, o aumento da profundidade da rede através do acréscimo de camadas convolucionais, além do uso de filtros 3X3 e funções de ativação ReLU nas camadas não lineares. Portanto, de modo experimental, os autores implementaram seis diferentes combinações, variando a quantidade de camadas de 11 até 19 e analisando os resultados obtidos por cada uma das composições.

Para a competição ILSVRC 2014, a arquitetura proposta, que possuía 19 camadas, atingiu taxa de erro top-5 de 7,3% nos testes.

2.3.5 GoogLeNet

Pode se definir a rede neural GoogLeNet, também conhecida por Inception-v1, como uma das particulares implementações da arquitetura Inception. Para essa construção em específico, tem-se uma composição de 27 camadas, além da utilização de 12 vezes menos parâmetros que a rede AlexNet e apresentar melhor taxa de acerto (SZEGEDY et al., 2015). Desse modo, não é somente o aumento no número de camadas que possibilita que um determinado modelo exerça sua função de forma mais performática que outro, mas vários fatores como sinergia em arquiteturas profundas e visão computacional clássica, como comentado por Szegedy em sua pesquisa.

O conceito por trás da arquitetura Inception baseia-se na busca de um ótimo local, de modo que essa estrutura possa ser reproduzida inúmeras vezes e o modelo construído pela agregação desses blocos convolucionais, chamados de *inception modules*.

A representação gráfica da rede neural pode ser encontrada no estudo de Szegedy et al. (2015), já que, por ser um modelo com grande quantidade de camadas, concluiu-se não ser viável sua exposição nessa seção. Durante a competição ILSVRC 2014, o modelo foi treinado com 1,2 milhões de imagens, validado com 50.000 e testado com 100.000, todas disponibilizadas pelo *dataset* ImageNet. Para esses casos, GoogLeNet antingiu uma taxa de erro top-5 de 6,67%.

2.3.6 ResNet

Destacando o marco no ramo de classificação de imagens proporcionado pelas redes neurais convolucionais, He et al. (2015) cita estudos onde torna-se evidente a influência da profundidade e quantidade de camadas para o desempenho dessas redes. Porém, quando uma rede *deep learning* começa a convergir, um problema de degradação tem certo potencial em ocorrer: o aumento da profundidade da rede pode fazer com que a acurácia fique saturada e então se degrade rapidamente e, como pontuado por He et al. (2015), o aumento no número de camadas, nesse caso, somente tende a aumentar o erro de treinamento. Desse modo, essa degradação mostra que nem todos os sistemas são fáceis de otimizar e a solução encontrada pelos pesquisadores foi o modelo ResNet.

Basicamente, o modelo funciona de maneira que, ao invés de esperar com que o valor de entrada, em um conjunto de camadas, seja diretamente mapeado para uma saída a partir de alguma função matemática, é buscado atingir um valor residual, isto é, se a saída desejada é H(x), então o modelo buscará otimizar F(X) = H(x) - x. Assim, além de ser considerado mais simples otimizar uma função já conhecida, é possível retornar para o valor desejado de forma elementar, sabendo que H(x) = F(x) + x. Ademais, a formulação dessa equação pode ser feita a partir de uma *feedforward* do valor de entrada (HE et al., 2015).

Tomando como base a competição ILSVRC 2015, os autores do artigo implementaram uma rede neural residual com 152 camadas, alcançando a taxa de erro top-5 de 4,49%, além de uma taxa top-5 de 3,57%, quando realizado a construção da arquitetura a partir da combinação de seis modelos com diferentes profundidades.

2.3.7 Xception

A arquitetura da rede neural Xception é construída em cima da ideia de separação das camadas convolucionais por profundidade, conceito semelhante ao introduzido pelos módulos *inception* no estudo de Szegedy et al. (2015). Dessa maneira, tem-se um modelo composto por 36 camadas convolucionais empilhadas através de uma estrutura linear com conexões residuais. Além disso, a implementação da rede foi elaborada utilizando o *framework* TensorFlow e treinada utilizando 60 GPUs de modelo NVIDIA K80 (CHOLLET, 2017).

Para o *dataset* ImageNet, o modelo obteve uma taxa de acerto top-1 de 79,0% e top-5 de 94,5%, em experimentos que demoraram cerca de três dias cada. Diferentemente das arquiteturas apresentadas anteriormente, o estudo de Chollet (2017) expressou a performance da rede através da acurácia e não do erro, comparando e sobressaindo em relação aos padrões VGG-16, ResNet-152 e Inception V3.

2.3.8 EfficientNet

Durante sua pesquisa, Tan e Le (2019) buscaram demonstrar como normalmente são desenvolvidos novos modelos de redes neurais convolucionais, começando com a criação de determinado padrão, que então é escalável, para obter melhores taxas de acerto variando atributos como profundidade e largura, por exemplo. Com isso, os autores propuseram uma maneira de realizar esse redimensionamento de forma uniforme, utilizando determinados coeficientes de escalonamento, e assim, a criação de um novo conjunto de arquitetura para CNN chamado de EfficientNet.

Uma das particulares implementações do modelo, conhecido por EfficientNet-B7, atingiu taxa de acerto top-1 de 84,3% e top-5 de 97,0% para o *dataset* ImageNet (TAN; LE, 2019).

2.4 Modelos Utilizando TensorFlow e Keras

Para representar os modelos de redes neurais virtualmente, a biblioteca de código aberto Python chamada Keras apresenta três alternativas: uma classe chamada *Sequential*, onde as camadas são conectadas em uma corrente única; uma *Functional* API (*Application Programming Interface*), que permite a representação dos modelos de redes como gráficos

acíclicos dirigidos, onde as camadas são os vértices e as conexões entre elas são as arestas; e por fim o *Model subclassing*, onde é possível criar modelos de camada não lineares, adicionando comportamentos arbitrários ao lado das chamadas das funções matemáticas já implementadas na biblioteca.

A classe *Sequential* se comporta como uma lista encadeada simples, onde as camadas são adicionadas uma após a outra, começando pela camada de entrada, sem qualquer ramificação ou realimentação, e passando por todas as camadas até terminar na camada de saída. Apesar de ser um modelo mais simples de se manipular, a rede fica restrita à apenas uma entrada e uma saída, o que pode ser uma limitação para a solução de alguns problemas.

Com a *Functional* API, é possível fazer com que a saída de uma camada se conecte à entrada de múltiplas camadas, possibilitando ramificações da rede. Também é possível conectar a saída de múltiplas camadas à entrada de uma única camada, possibilitando a convergência de ramificações em um único ponto. Este tipo de estrutura é chamado de Grafo Acíclico Dirigido (GAD), onde, para cada vértice no grafo, há um número finito de caminhos que partem dele (arestas de saída), e todos levam à um vértice final (que não possui saídas). Diferente do modelo *Sequential*, com a *Functional* API é possível ter múltiplas entradas, múltiplas saídas, e camadas que não são influenciadas diretamente pelo resultado de outras.

Com o *Model subclassing*, a biblioteca abre o leque para a realimentação de camadas, operações aritméticas com partes da rede, entre outros comportamentos não lineares. A ideia se baseia no polimorfismo da Programação Orientada à Objetos (POO), onde um objeto pode generalizar o comportamento de outro e adicionar novos comportamentos. Neste caso, um novo tipo de camada poderá emular o comportamento de outra já nativa da biblioteca, adicionando ou removendo etapas do processo original.

Além disso, a biblioteca conta com uma API de aplicações, onde estão disponíveis diversas redes, inclusive algumas das abordadas na seção 2.3. É possível carregá-las de maneira pré-treinada, ou seja, prontas para serem utilizadas em classificações e reaproveitadas em novos treinamentos, ou até sem treinamento inicial: com pesos definidos de forma aleatória para que um treinamento seja realizado do zero.

2.4.1 Funções de Perda e Otimização

Com a biblioteca Keras, após a definição da arquitetura de uma aplicação de rede neural, é preciso escolher alguns parâmetros de treinamento, sendo eles: a função de perda; a função de otimização; o tamanho do lote de arquivos que será considerado em cada atualização dos pesos; e o número de vezes que a rede irá iterar sob todo o conjunto de dados.

Definir a função de perda é definir qual função matemática o treinamento deve minimizar. Ela é a função que compara o resultado gerado pela rede com o resultado dos dados escolhidos para treino. Essa comparação gera um valor numérico, que será usado pela função de otimização para recalcular o valor dos pesos da rede. Desta maneira, pode-se dizer que quanto maior o valor obtido na função de perda, mais distante a rede está de seu estado ideal e, portanto, sofrerá maior alteração nos valores de seus pesos.

A escolha da função de perda deve ser feita de acordo com cada problema. Em Chollet (2018, p. 114), o autor traz uma tabela para orientar a escolha da função de acordo com alguns casos comuns (Tabela 1).

Tabela 1 – Funções de perda

Tipo de Problema	Função de Perda
Classificação Binária	Entropia Cruzada Binária
Classificação Multi classe, Rótulo Único	Entropia Cruzada Categórica
Classificação Multi Classe, Múltiplos Rótulos	Entropia Cruzada Binária
Regressão para valores arbitrários	Erro Quadrado Médio
Regressão para Valores entre 0 e 1	Erro Quadrado Médio ou Entropia Cruzada Binária

Fonte: Adaptado de Chollet (2018).

A função de otimização é o que determina a maneira como os pesos serão atualizados, com base no resultado da função de perda. Suas implementações são variantes específicas da função matemática *Gradiente Descendente Estocástico*. Ela define, para cada peso da rede, uma alteração na tentativa de minimizar o resultado da função de perda (CHOLLET, 2018, p. 11 e 29).

Chollet (2018, p. 113) diz que, para a maioria dos casos, é seguro escolher o método RMSProp, porém, a API Keras disponibiliza outras sete funções de otimização, que são: SGD, Adam, Adadelta, Adagrad, Adamax, Nadam e Ftrl. A definição das funções vem acompanhada da escolha de um parâmetro, sua taxa de aprendizado. Na mesma página, o autor também cita que, alguns destes modelos de função utilizam o conceito de taxa de momento de inércia como um segundo parâmetro. Defini-lo implica que o valor da função de perda utilizado em cada rodada de otimização será influenciado não só pelo valor computado naquele lote, mas também pelo valor dos lotes anteriores.

2.4.2 Transferência de Aprendizado

A técnica denominada *transfer learning* é amplamente empregada quando deseja-se utilizar de aplicações de redes neurais já treinadas para classificar *datasets* próprios. Assim, é possível aproveitar dos pesos de um determinando modelo com acurácia e classificações conhecidas, encurtando assim o tempo de treinamento e diminuindo o poder

de processamento necessário, uma vez que o treinamento da nova aplicação *deep learning* não irá começar do zero (KRISHNA; KALLURI, 2019). Ademais, essa técnica pode ser implementada utilizando-se das funcionalidades das bibliotecas TensorFlow e Keras.

2.5 Versionamento de Modelos Treinados de Redes Neurais

Redes neurais profundas, ou *Deep Neural Networks* (DNN), trouxeram uma série de avanços no estado da arte. Entretanto, ainda há pontos críticos no gerenciamento do ciclo de vida de modelos de *deep learning* que tem sido ignorados por pesquisadores atualmente. O sistema ModelHub, apresentado por Miao et al. (2017), foi criado com o intuito de lidar com as tarefas ligadas ao uso de redes neurais de uma maneira mais compreensível. O ModelHub apresenta um sistema de versionamento de modelos (DLV), um gerenciador de linhagem, uma linguagem para consultas (DQL) e serviços para armazenar, explorar e compartilhar modelos. O foco está no modelo de versionamento presente neste sistema, pois pelo alto custo e tempo da fase de treinamento dos modelos, é necessário ter diversos *snapshots* ou *checkpoints* para salvar o estado do modelo em execução. O sistema de versionamento DLV é implementado como uma ferramenta de linha de comando que serve como interface para outros componentes como navegar e buscar por, artefatos gerados no ciclo de vida do modelo, definições de rede, registros de treinamento, relacionamentos, entre outros. Uma versão de modelo consiste em uma definição da rede, uma coleção de pesos, de parâmetros, metadados e arquivos usados em conjunto. Ademais, uma nova versão é salva quando o usuário executa um comando no DLV que altera o repositório.

2.6 Armazenamento e Versionamento de Dados em Larga Escala

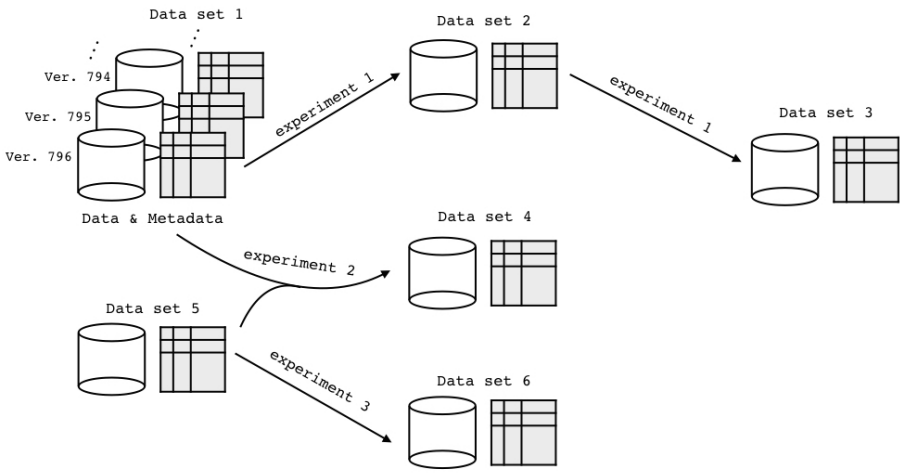
O crescimento da *internet*, *smartphones* e sensores sem fio tem produzido uma enorme diversidade de conjuntos de dados em todos os aspectos de nossas vidas. Os conjuntos de dados são variados em sua natureza, pequenos ou grandes, estruturados ou desestruturados. Pesquisadores e cientistas de dados possuem o interesse em coletar, analisar e trabalhar de forma colaborativa para extrair um conhecimento através dos dados. Entretanto, os sistemas atuais não são adequados para este tipo de ambiente colaborativo. Os bancos de dados relacionais funcionam bem quando as informações estão devidamente estruturadas, mas não apresentam um gerenciamento de versão e, caso seja utilizado um sistema de versionamento, como o *git*, uma série de problemas envolvendo conjuntos de dados grandes podem ocorrer, levando times a armazenarem os dados em sistemas de arquivos e gerenciar versões manualmente (BHARDWAJ et al., 2014).

Bhardwaj et al. (2014) propõe dois sistemas, o primeiro chamado de Data Version Control System (DSVC), o qual tem a função de salvar modificações, utilizar linguagens declarativas nas versões, comparar diferenças entre versões e possibilitar a analise colabo-

rativa. O DSVC controla conjuntos de dados enormes e a eficiência da ferramenta reside no uso de algoritmos eficientes de identificação e comparação de diferenças entre versões, decidindo quais versões devem ser preservadas completamente e outras comprimidas. O segundo sistema consiste em uma plataforma hospedada no topo do DSVC, chamado de DataHub, o qual oferece uma série de ferramenta para o tratamento de dados.

O DataLab foi criado com o intuito de gerenciar a revisão de códigos e dados em um sistema. O seu funcionamento reside no controle dos resultados da análise de dados em um grafo de fluxo de trabalho, que permite visualizar diferenças entre duas versões, como representado na Figura 7. O armazenamento é feito de forma que os metadados são separados, permitindo o uso de filtros de pré-processamento eficientes. Em relação a códigos inseridos por usuários, o DataLab utiliza elementos da API GitLab para gerenciamento, como o *commit ID* gerado para mudanças. Para tornar eficiente a busca, são armazenadas etiquetas e anotações de cada arquivo para serem buscados eficientemente em um banco de dados NoSQL. Os dados e metadados são vinculados por meio de um identificador único gerado pela plataforma. O projeto utiliza o MongoDB como banco de dados, com plano de migração para um sistema orientado a colunas para melhorar ainda mais a performance (ZHANG et al., 2016).

Figura 7 – Exemplo de um grafo de fluxo de trabalho



Fonte: Zhang et al. (2016)

O projeto Machine Learning data platform (MLdp) oferece um modelo de dados minimalista e flexível para todos os tipos de dados, um forte gerenciamento de versão para garantir reprodutibilidade dos experimentos de aprendizado de máquina, além de suportar integração com diversos *frameworks* de *machine learning*. O MLdp é mais do que um armazém de dados, o objetivo é auxiliar tarefas de *machine learning* como anotação, exploração, engenharia de dados, e outras funcionalidades. A plataforma armazena os dados em colunas, com o usuário podendo definir o caminho de acesso, como índices primários, secundários, índices parciais, etc. Tudo isso permite uma exploração de dados

e filtragem eficiente ao sistema. O versionamento no MLdp é explícito e controlado pelo portador dos dados. É um gerenciamento de versão que permite: compartilhar e evoluir versões conforme sua cadência e necessidade sem interromper outros projetos; fixar uma versão específica para reproduzir resultados de treinos; ou até rastrear dependências de versões entre dados e modelos treinados (AGRAWAL et al., 2019).

2.7 Ferramenta para o Desenvolvimento de Modernas GUI

O ReactJS foi desenvolvido pelo Facebook, atualmente conhecido como Meta. Essa ferramenta pode ser entendida como uma biblioteca JavaScript de código aberto para o desenvolvimento *front-end* e que pode ser instalada em diferentes ambientes, como MAC OS, Windows e UNIX. Ela é amplamente utilizada para a construção de modernas interfaces de usuário com componentes reutilizáveis e que prezam pelo estilo *Single Page Application* (SPA), isto é, páginas que não necessitam de um novo *reload* a cada clique do usuário. Segundo Rawat e Mahajan (2020), uma das principais características dessa biblioteca é o fato dela não ser muito complicada de se utilizar e aprender, fazendo com que o programador gaste menos tempo com o desenvolvimento da interface gráfica em si e possa focar em outras questões, como regras de negócio, por exemplo.

Outro ponto importante relacionado à construção de *Graphical User Interfaces* (GUI) usando o ReactJS é o fato de cada componente possuir seu próprio estado e ciclo de vida, fazendo com que seja possível a manipulação de dados a partir do *data binding*, além de possibilitar com que alterações no *Document Object Model* (DOM) ocorram praticamente de forma automática após o retorno de um *request* à uma API, por exemplo. Além disso, o ReactJS possui suporte a diferentes pacotes que estão disponíveis no *Node Package Manager* (NPM) e que podem auxiliar no desenvolvimento de uma aplicação. Alguns desses pacotes são o React Router, responsável por lidar com o roteamento de páginas dinâmicas; os pacotes Material UI e Bootstrap, que podem prover estilosos componentes para a aplicação e o pacote Axios, que é muito utilizado para lidar com requisições do *Hypertext Transfer Protocol* (HTTP) (RAWAT; MAHAJAN, 2020).

Um dos principais fatores que implicam na escolha dessa ferramenta, para o desenvolvimento de uma aplicação *front-end*, é a performance. Enquanto algumas das bibliotecas e *frameworks*, utilizados para o mesmo fim, interagem diretamente com a árvore do DOM a cada necessidade de modificação na página, o ReactJS mantém uma espécie de cópia deste documento em memória, conhecido por virtual DOM e o que torna o acesso à esta árvore ligeiramente mais rápido. Desse modo, sempre que é necessário a correlação entre as duas partes, isto é, do modelo virtual para o modelo real, um algoritmo *diff* é empregado e apenas as partes com alterações são refletidas (BHALLA; GARG; SINGH, 2020).

Outrossim, como apresentado por Bhalla, Garg e Singh (2020) em seu estudo, o ReactJS é construído basicamente em cima da camada *View*, presente no modelo MVC (*Model View Controller*). Isso permite com que a aplicação seja dinâmica, a partir do controle de eventos, mas que seja necessário outros subprogramas e abordagens, tais como os pacotes adicionais exemplificados por Rawat e Mahajan (2020), para que em conjunto, possam implementar completamente o projeto proposto.

2.8 Trabalhos Relacionados

Nesta seção são descritos outros *softwares* desenvolvidos com o mesmo objetivo: diminuir a barreira de conhecimento necessária para criação, treinamento e uso de aplicações de *machine learning* e, portanto, democratizar o acesso à essa tecnologia. Em particular, serão citados os *softwares*: DVC, desenvolvido pela empresa de mesmo nome; Edge Impulse, também desenvolvido pela empresa de mesmo nome; SciANN, desenvolvido por pesquisadores; CreateML, desenvolvido pela Apple; AutoML, desenvolvido pela Google; e Neuro-T, desenvolvido pela empresa Neurocle.

2.8.1 DVC

Data Version Control (DVC), é uma ferramenta *open-source* de gerenciamento de experimento que controla dados e modelos de redes neurais para aproveitar de outras ferramentas de engenharia na qual são largamente utilizados. Atualmente possui as funções de rastrear e salvar dados e modelos de *machine learning*, criar e trocar entre versões de dados e *machine learning* facilmente, além de adaptar ferramentas e boas práticas para projetos. A importância do DVC está no trabalho em conjunto com o *Git*, permitindo realizar as mesmas operações que são conhecidas na ferramenta, o que traz um ponto essencial para experimentos, a reprodutibilidade (DVC, 2022). A ferramenta DVC não só versiona modelos de *machine learning*, como versiona grande volume de dados por meio de pequenos metadados com a ferramenta *Git*. Os dados e modelos são armazenados em nuvens ou servidores, permitindo o compartilhamento de grandes *datasets* ou modelos treinados em GPU. O DVC é majoritariamente voltado para programadores, por necessitar de um conhecimento maior na área, enquanto o DeepUAI foca no trabalho de tornar o uso de redes neurais acessível por meio de uma interface gráfica, podendo ser utilizado por um público menos experiente.

2.8.2 SciANN

Uma outra aplicação é apresentada por Haghighat e Juanes (2021), chamada de SciANN (*Scientific Computing with Artificial Neural Networks*). SciANN é uma biblioteca *open-source* de redes neurais baseada no Tensorflow e Keras, também utilizados no DeepUAI.

A abstração utilizada na interface de programação tem como objetivo suportar aplicações de engenharia, como *model fitting* e *model inversion*, além de soluções de equações diferenciais ordinárias e parciais. O artigo apresenta em detalhes a utilização do pacote para resolução de diversos problemas, é um conteúdo focado fortemente em resoluções matemáticas. Assim, a biblioteca SciANN tem o mesmo intuito de facilitar o acesso ao uso de *deep learning* e redes neurais, porém ainda necessita de uma familiaridade com a linguagem Python, visto que não apresenta uma interface gráfica amigável ao usuário, ela é um pacote que oferece abstrações dentro da linguagem. O grande foco na resolução de problemas matemáticos também é um diferencial em relação ao DeepUAI.

2.8.3 Edge Impulse

A plataforma de desenvolvimento de *machine learning* Edge Impulse possibilita que o usuário "construa habilidades de *machine learning* embarcado através do tempo com a plataforma, que pode ser usada como uma ferramenta de pouco ou nenhum código, até um Jupyter Notebook que pode ser usado para fazer chamadas à API da plataforma"(Edge Impulse, 2022b, *tradução livre*) [1]. A plataforma também utiliza da ferramenta Keras e cobre todo o fluxo de trabalho necessário para a criação de modelos de *machine learning*, desde a criação de conjuntos de dados, classificação dos conjuntos dados, escolha de modelos de redes neurais e/ou outros algoritmos de aprendizado de máquina e processamento de sinais digitais, treinamento dos modelos, uso dos modelos treinados para inferências e implantação destes modelos em diversos tipos de dispositivos. A principal diferença entre as plataformas Edge Impulse e DeepUAI está no foco de seu uso, pois a primeira possibilita a criação de aplicações de ML para diversos tipos de tarefas, não só de visão computacional, e, mesmo que disponibilizando ferramentas de pouco ou nenhum código, está focada no desenvolvedor, trazendo total transparência e poder de configuração à todas as etapas do processo e utilizando do slogan "De desenvolvedores, para desenvolvedores"(Edge Impulse, 2022a, *tradução livre*) [2], enquanto a segunda tem como foco usuários não programadores, com o objetivo de possibilitar que com um clique ou dois o usuário crie o próprio modelo, escondendo algumas etapas do processo.

2.8.4 CreateML

Assim como a plataforma Edge Impulse, a aplicação da Apple CreateML traz uma interface onde é possível realizar todo o processo de criação de uma aplicação de rede neural para diversos tipos de tarefas, para além da visão computacional, porém com a diferença que o usuário não escolhe a arquitetura da rede, apenas qual tarefa ela irá realizar, deixando a escolha da arquitetura internalizada dentro do *software*. Também é

[1] Build in-house embedded ML skills over time with Edge Impulse, which can be accessed as a low-to-no code tool to a Jupyter notebook which can be used to make API calls to the platform

[2] From developers, for developers

uma ferramenta voltada para desenvolvedores, possibilitando a customização dos modelos pré-definidos pela aplicação através de alterações no código da biblioteca para *machine learning* desenvolvida pela própria empresa, na linguagem Swift. É possível utilizar as redes treinadas com o CreateML para avaliação de dados, mas diferentemente da ferramenta Edge Impulse, o CreateML não aborda a etapa de implantação do modelo em dispositivos, pois essa funcionalidade é disponibilizada por outra ferramenta, também da Apple, a CoreML. A principal diferença entre a CreateML e a DeepUAI é, além das mesmas encontradas entre a DeepUAI e a Edge Impulse, que o usuário fica restrito a dispositivos com sistemas operacionais da empresa, como macOS e iOS, enquanto a DeepUAI traz uma abordagem mais democrática e independente de sistemas operacionais.

2.8.5 Neuro-T

O Neuro-T faz parte de um conjunto de produtos voltados para desenvolvimento de *deep learning*, em que ele é responsável pela etapa de criação e treinamento das redes neurais através de interfaces, sem necessidade de construção de códigos pelo usuário, deixando a etapa de implantação em dispositivos para a ferramenta Neuro-X, também da mesma empresa, em um paralelo com as aplicações CreateML e CoreML da Apple. Assim como a CreateML, a Neuro-T esconde a etapa de escolha da arquitetura com o que ela chama de "Auto Deep Learning Algorithm". Segundo a empresa, "Auto Deep Learning Algorithm automaticamente otimiza fatores como o modelo de rede, otimizadores, métodos de decaimento, taxa inicial de aprendizado, tamanho de lotes e épocas para treinamento otimizados de modelos. Com esse algoritmo, qualquer um pode usufruir da tecnologia deep learning"(Neurocle, 2022, *tradução livre*) [3]. A principal diferença entre a Neuro-T e a DeepUAI é a possibilidade que o DeepUAI permite escolher o modelo de rede neural que será utilizado na aplicação, diferente da Neuro-T que escolhe automaticamente. A Neuro-T trabalha apenas com aplicações para visão computacional, diferentes das ferramentas Edge Impulse e CreateML, que são de propósito geral, além de não possibilitar personalizações via código.

2.8.6 AutoML

A Google, empresa responsável pela ferramenta AutoML, diz que: "O AutoML promove a expansão de empresas e indivíduos com capacidade de se beneficiar da IA, oferecendo uma experiência do usuário sem código e fácil de usar, que não requer nenhum conhecimento prévio em *machine learning*"(Google Cloud, 2022). A ferramenta, assim como a Edge Impulse e a CreateML, pode ser usada para construção de aplicações de *machine learning* para diversos propósitos, não se limitando à visão computacional, e

[3] Auto Deep Learning Algorithm automatically optimizes all factors such as the model network, optimizer, decay method, initial learning rate, batch size, and epoch for optimal model training. With this algorithm, anyone can take advantage of deep learning technology

também possibilita que todo o processo de criação de uma aplicação de *machine learning* seja concluído sem nenhum uso da programação. Além disso, o AutoML disponibiliza ferramentas para que os usuários personalizem os modelos na ferramenta com inserção de código. Ela também conta com as ferramentas de outros serviços da empresa para implantação dos modelos, o Firebase Machine Learning, que é uma ferramenta voltada para programadores disponibilizarem suas aplicações de ML na nuvem e em dispositivos móveis. A principal diferença entre a DeepUAI e a AutoML é a possibilidade de seleção dos modelos via interface, pois na AutoML o modelo é escolhido a partir de configurações dadas pelo usuário sobre preferências de performance e tamanho da rede em disco.

A comparação destes trabalhos relacionados com o proposto, denominado DeepUAI, pode ser observada na Tabela 2.

Tabela 2 – Comparação entre trabalhos relacionados e suas características

Aplicações	α	β	γ	δ	ϵ	ζ	ϕ
DeepUAI	X	X	X	X	X		
DVC		X	X			X	X
SciANN				X			X
Edge Impulse	X	X	X	X	X	X	X
CreateML	X					X	X
Neuro-T	X				X		
AutoML	X				X	X	X

α: Interface gráfica; β: Permite a escolha da arquitetura; γ: Versionamento de aplicações; δ: Utilização das bibliotecas Tensorflow e/ou Keras; ϵ: Não requer conhecimento em programação; ζ: Diversidade de aplicação; ϕ: Pode ser explorado com programação.

Fonte: Elaborado pelos autores com base em DVC; SciANN; Edge Impulse; CreateML; Neuro-T e AutoML , 2022.

3 Desenvolvimento

O desenvolvimento do *software* proposto foi baseado nos seguintes passos: prototipação inicial da interface gráfica, estudo dos casos de uso e fluxos da aplicação, mapeamento das entidades, definição da arquitetura, escolha das ferramentas a serem utilizadas e implementação das funcionalidades apresentadas.

3.1 Prototipação da Interface Gráfica

A interface gráfica é um elemento essencial para que o usuário consiga interagir com o sistema por meio de menus e botões. O sistema DeepUAI, como dito anteriormente, tem como objetivo facilitar o uso de aprendizado de máquina, portanto, uma interface gráfica amigável ao usuário é essencial para uma boa experiência.

A fim de visualizar graficamente o *design* da aplicação a ser desenvolvida antes de sua implementação em código, foi utilizada a ferramenta de prototipação Figma[1] e um modelo de interface gráfica de contexto geral para inspiração, disponibilizado pela comunidade[2].

Basicamente, essa ferramenta permite profissionais da área de tecnologia construir interfaces para produtos digitais, como sites e aplicativos, através de um *software* hospedado diretamente na *internet*, ou seja, sem a necessidade da instalação em máquina.

3.2 Casos de Uso e Fluxos da Aplicação

A Figura 8 descreve as funcionalidades que o sistema disponibiliza para os usuários e os processos que são realizados no servidor a partir dessas ações.

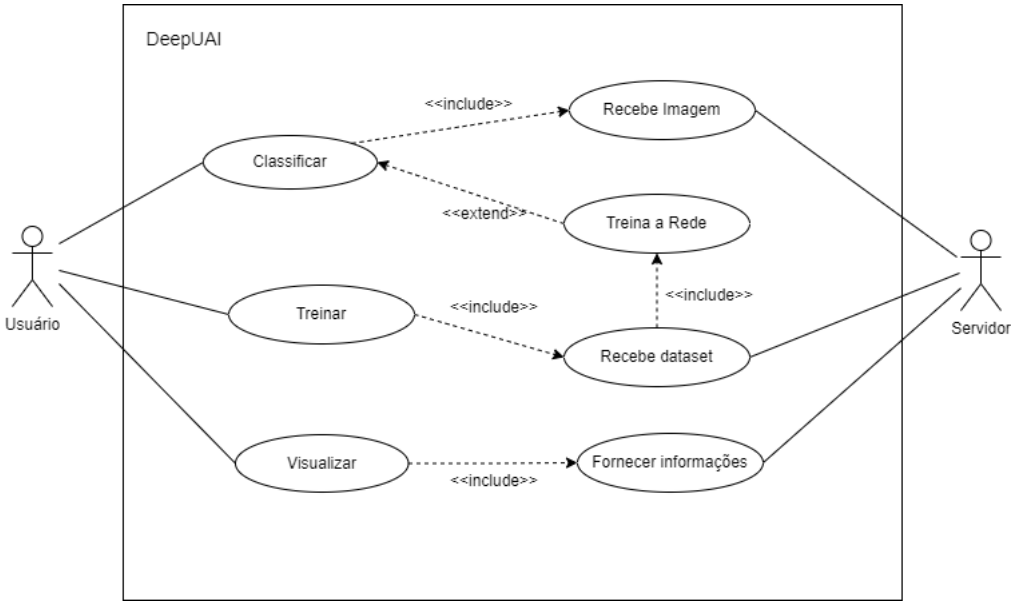
Como descrito no diagrama da Figura 8, o usuário pode visualizar uma aplicação de rede neural, classificar uma imagem qualquer com essa rede, ou treiná-la com novos dados, o que implicará em ações no servidor, por exemplo, receber a imagem a ser classificada e devolver as predições realizadas pela aplicação de rede neural em questão.

Além disso, ao escolher treinar uma rede, o treinamento será processado pelo sistema e, ao final dele, essa nova aplicação ficará disponível com todas as mesmas funcionalidades. Desse modo, as funções disponíveis no sistema são as seguintes:

[1] Disponível em <https://www.figma.com/>
[2] Disponível em <https://www.figma.com/community/file/1106591067539499223>

- Utilização de redes neurais convolucionais treinadas para classificar imagens importadas pelo usuário;
- Treinamento de redes neurais, previamente disponíveis, a partir de um *dataset* qualquer fornecido;
- Visualização de informações de determinado modelo de rede neural e sua respectiva versão, a fim de identificar o histórico da aplicação de inteligência artificial;
- Inserção e visualização dos conjuntos de dados utilizados para realizar o treinamento dos modelos.

Figura 8 – Diagrama de Casos de Uso



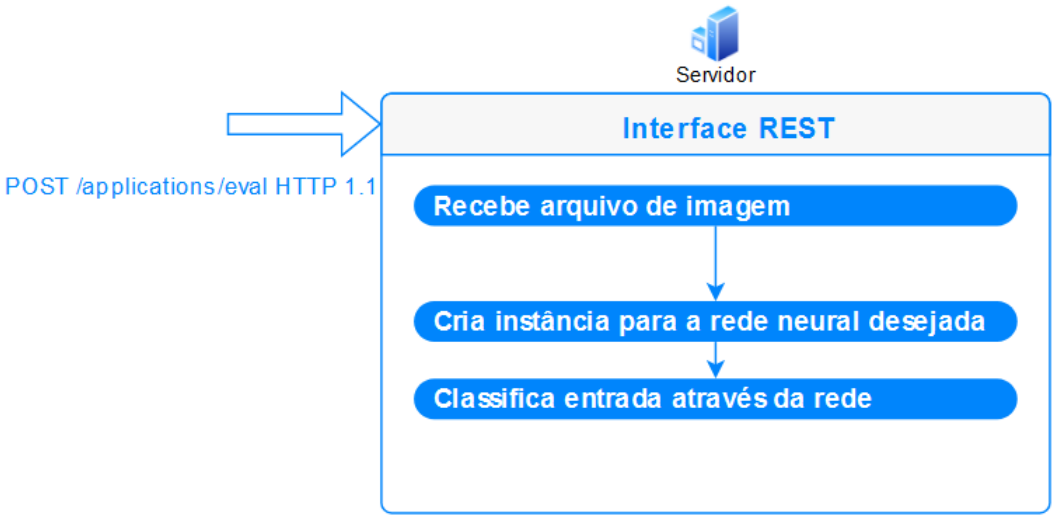
Fonte: Autoria própria, 2022.

Os dois principais fluxos do sistema são: classificação e treinamento. A funcionalidade de classificação foi pensada seguindo inteiramente o protocolo HTTP, uma vez que, o processo de predição para um único arquivo de entrada tende a não ser muito demorado, fazendo com que o usuário não aguarde a resposta desejada por um longo período de tempo. Assim, uma imagem é enviada através de um *request* do tipo POST ao servidor, juntamente com as informações necessárias da rede neural, e as três primeiras classificações para o dado de entrada são retornadas (Figura 9).

Já para a parte relacionada ao treinamento de redes neurais convolucionais, tornou-se necessário a utilização de um sistema de filas, de modo a priorizar e organizar as solicitações de treinamento, além de possibilitar que aconteçam de forma externa à requisição HTTP, pois um treinamento pode durar dias para ser finalizado. Desse modo, o usuário envia o *dataset* desejado para treinamento da rede, em conjunto com as informações da arquitetura escolhida, através de uma requisição do tipo POST para o servidor, que

publica uma mensagem na fila, finalizando a requisição e devolvendo ao usuário a informação de que este processo está aguardando para ser executado por outro serviço do servidor, que mantém-se ativo em *loop*, consumindo as mensagens da fila e realizando os respectivos treinamentos (Figura 10). Para isso, empregou-se o *software* RabbitMQ[3], visando sua funcionalidade de mensageria em filas.

Figura 9 – Fluxo da funcionalidade de classificação



Fonte: Autoria própria, 2022.

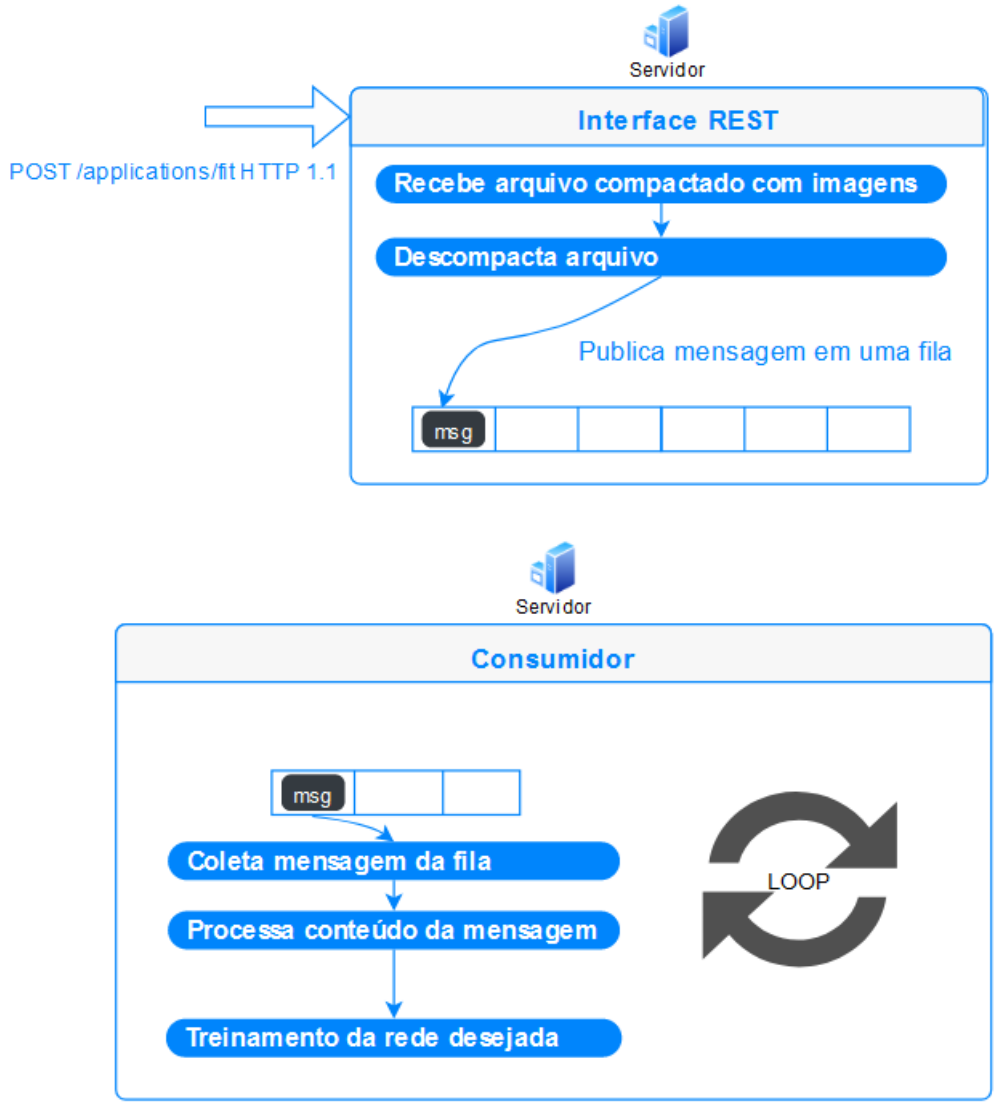
3.3 Mapeamento das Entidades

A persistência dos dados e metadados necessários para o funcionamento do sistema podem ser distribuídos em três entidades: *Model*, *Dataset* e *Application*. Como os nomes já sugerem, a primeira é responsável por armazenar as informações dos modelos de redes neurais, a segunda dos conjuntos de dados, e a terceira das aplicações *deep learning*, isto é, das arquiteturas convolucionais treinadas a partir de determinado conjunto de imagens. Além disso, existem três relacionamentos entre as entidades, já que uma aplicação é construída em cima de um modelo, treinada a partir de um *dataset* e ainda pode ser construída a partir de outra aplicação já existente, utilizando-se da técnica denominada transferência de aprendizado (Figura 11).

Para o armazenamento de dados escolheu-se a utilização de um banco de dados objeto relacional, tendo em vista o aparente relacionamento entre as entidades na Figura 11. Desse modo, o sistema utilizado é o PostgreSQL[4], devido a este ser um dos bancos de dados *open source* mais utilizados na comunidade de desenvolvimento de *software*.

[3] Disponível em <https://www.rabbitmq.com>
[4] Disponível em <https://www.postgresql.org>

Figura 10 – Fluxo da funcionalidade de treinamento



Fonte: Autoria própria, 2022.

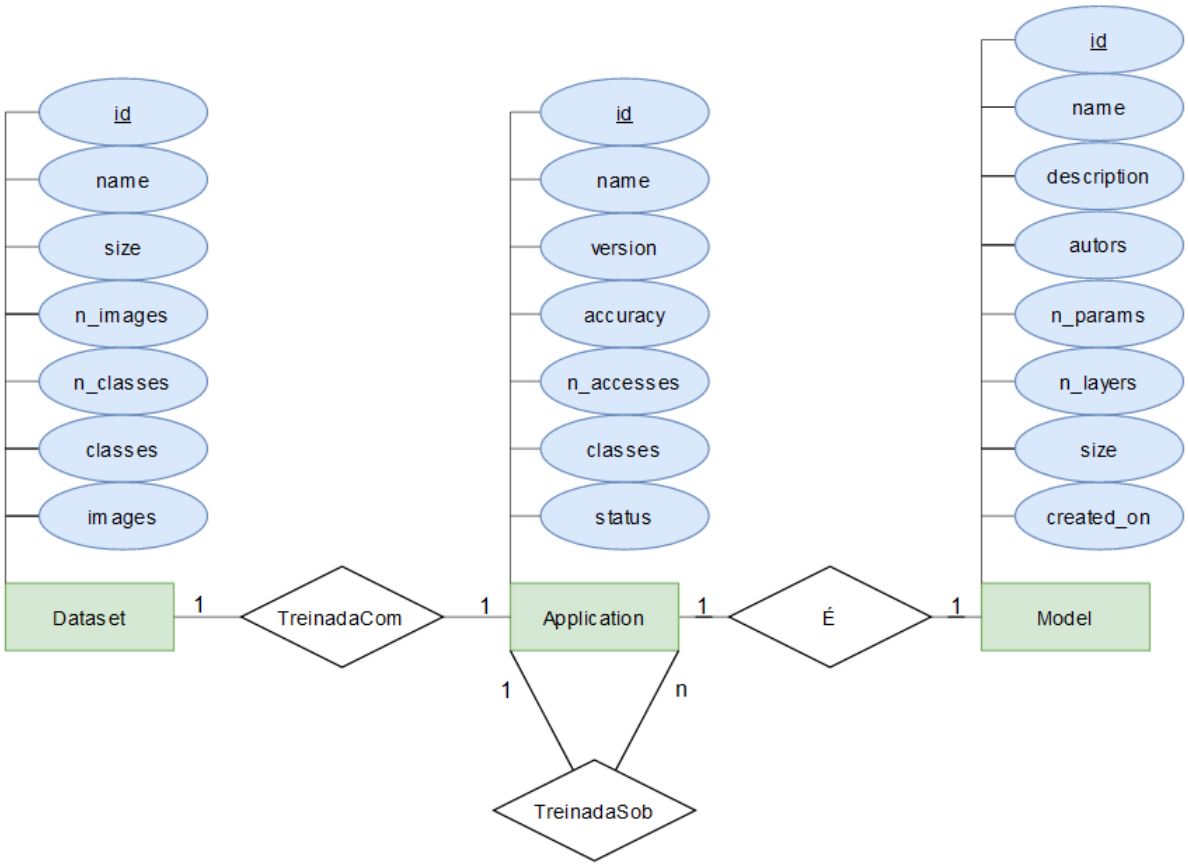
3.4 Arquitetura

O modelo arquitetônico da aplicação é composto por duas camadas: *front-end* e *back-end* (Figura 12).

No *front-end* encontra-se a interface gráfica da ferramenta, isto é, o local de interação para o usuário final da aplicação. Através dela ele poderá enviar seus conjuntos de dados e utilizar as funcionalidades do sistema.

Já no *back-end* existem três serviços: servidor, API e banco de dados. O servidor é o responsável por gerenciar o processamento relacionado às redes neurais artificiais e armazenar os *datasets* e as aplicações *deep learning*; o banco de dados armazena metadados referentes aos processos de treinamento e importação de *datasets* no banco de dados; e a API possui a função de entregar à aplicação gráfica as informações pertinentes ao escopo

Figura 11 – Diagrama Entidade e Relacionamento



Fonte: Autoria própria, 2022.

do sistema e que estejam armazenadas no banco de dados, como por exemplo listagem de aplicações *deep learning* e listagem de *datasets* disponíveis.

3.5 Ferramentas

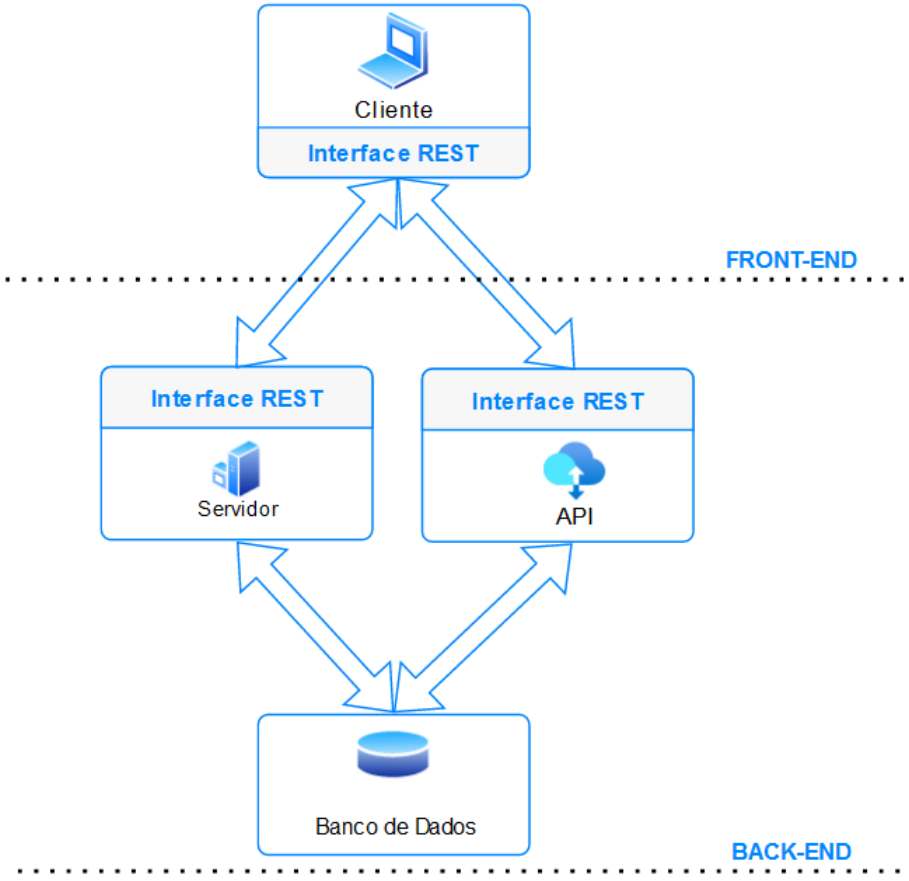
Para a construção do *front-end* utilizou-se da biblioteca ReactJS em JavaScript, com algumas bibliotecas auxiliares: React Bootstrap, Axios, react-chartjs-2 e Cytoscape.js[5]. A biblioteca React Bootstrap foi utilizada com o intuito de fornecer alguns componentes como botões, menus e formulários já estilizados; o cliente HTTP Axios para implementar a comunicação da GUI com o servidor e a API; o react-chartjs-2 para prover os componentes React relacionados a gráficos; e por fim o Cytoscape, visando a possibilidade de observar a versão de determinada rede neural como um grafo direcionado.

O desenvolvimento do servidor foi realizado em Python utilizando dos pacotes: TensorFlow, Pillow, NumPy, FastAPI, aiofiles, Psycopg 2 e Pika[6]. A biblioteca da Google

[5] Bibliotecas disponíveis respectivamente em: ReactJS: <https://reactjs.org>; React Boostrap: <https://react-bootstrap.github.io>; Axios: <https://axios-http.com>; react-chartjs-2: <https://react-chartjs-2.js.org>; Cytoscape.js: <https://js.cytoscape.org>.

[6] Bibliotecas disponíveis respectivamente em: TensorFlow: <https://www.tensorflow.org>; Pillow:

Figura 12 – Arquitetura do sistema DeepUAI



Fonte: Autoria própria, 2022.

TensorFlow, e de maneira especial a API Keras - a qual é construída a partir dessa plataforma - é empregada para a implementação das funções relacionadas à inteligencia artificial, como instanciação de modelos, criação de redes neurais, classificação e treinamento; já as bibliotecas Pillow e NumPy para realizar as manipulações necessárias nas imagens, a fim de executar a etapa de classificação para uma determinada rede neural; o pacote FastAPI para prover a interface REST ao servidor; aiofiles para o procedimento de salvar determinado arquivo compactado - representando certo conjunto de dados - que é recebido pelo servidor através de uma requisição HTTP; Psycopg 2 para a manipulação do banco de dados PostgreSQL; e a biblioteca Pika, a qual provê os recursos necessários para gerenciar o servidor de mensageria em fila utilizado, o RabbitMQ.

A API foi implementada em NodeJs[7], tendo em vista o emprego das seguintes bibliotecas e *frameworks*: Express, CORS, body-parser, consign, pg e Knex.js[8]. O *fra-*

<https://pillow.readthedocs.io>; NumPy: <https://numpy.org>; FastAPI: <https://fastapi.tiangolo.com>; aiofiles: <https://pypi.org/project/aiofile>; Psycopg: <https://www.psycopg.org>; Pika: <https://pika.readthedocs.io>

[7] Ambiente de execução JavaScript, disponível em <https://nodejs.org>

[8] Bibliotecas disponíveis respectivamente em: Express: <https://expressjs.com>; CORS: <https://www.npmjs.com/package/cors>; body-parser: <https://www.npmjs.com/package/body-parser>; consign:

mework Express foi aproveitado a fim de prover a interface HTTP, em conjunto com as bibliotecas CORS, body-parser e consign; Já as bibliotecas pg e Knex.js foram utilizadas associadamente visando a manipulação do banco de dados PostgreSQL através da API.

Além disso, as ferramentas Docker e Docker-compose também são utilizadas para executar o banco de dados PostgreSQL e o *software* de mensagens RabbitMQ.

3.6 Configuração do Treinamento

Para definição da função de perda, foi adotado como padrão a orientação da Tabela 1 para casos de classificação de múltiplas classes, a Entropia Cruzada Binária.

A função de otimização escolhida foi a Adam, com seus valores padrão nos parâmetros que controlam a magnitude das alterações feitas a cada iteração do treinamento.

Para o tamanho dos lotes de dados considerados em cada iteração também foi utilizado o valor padrão, de 32 imagens. O número de épocas (vezes que a rede treina sob todo o *dataset*) foi definido como cinco vezes.

<https://www.npmjs.com/package/consign>; pg: <https://www.npmjs.com/package/pg>; Knex.js: <https://knexjs.org>

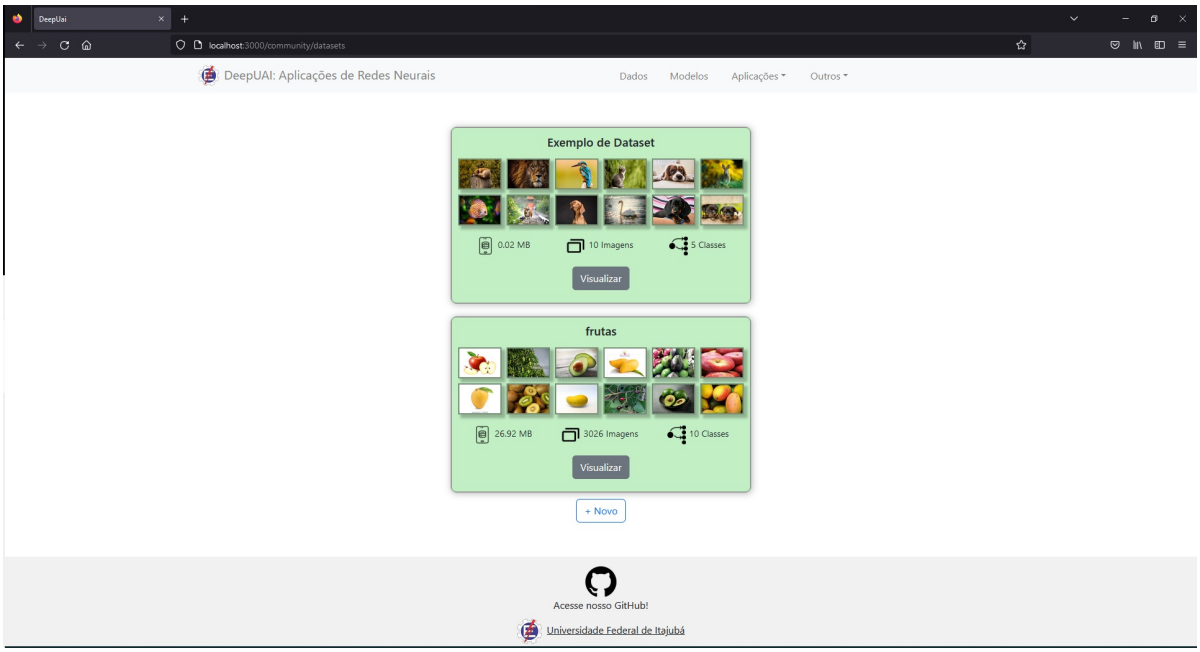
4 Resultados

Obteve-se como resultado uma aplicação *web* completamente funcional, organizada em um menu superior que apresenta quatro abas, sendo elas: *Dados*, *Modelos*, *Aplicações* e *Outros*. Nas três primeiras abas são contempladas as funcionalidades necessárias para que o usuário seja capaz de criar uma nova aplicação de rede neural sem o uso de nenhuma outra ferramenta, enquanto na última estão informações extras, como informações sobre os autores e o trabalho desenvolvido.

4.1 Dados

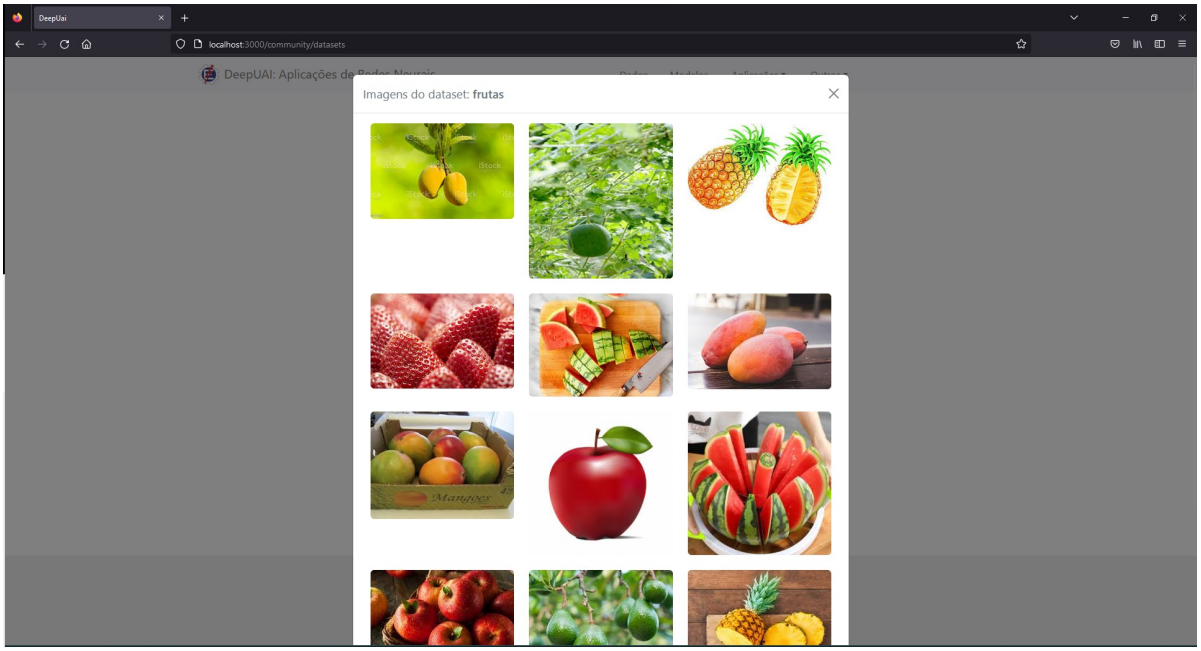
A primeira é a aba *Dados*, apresentada na Figura 13. Ela é composta de uma única página, onde são listados todos os conjuntos de dados disponíveis na plataforma. Essa página consiste de uma lista vertical, contendo um componente visual para cada conjunto de dados disponível. Cada componente de conjunto de dados contém um botão com o título *Visualizar*, que abre na tela a lista de todas as imagens do conjunto (Figura 14). Há também, no final da lista, um botão que possibilita que o usuário envie um novo conjunto de dados na forma de um arquivo compactado e disponibilize-o junto dos outros.

Figura 13 – Página com os conjuntos de dados disponíveis na aplicação DeepUAI



Fonte: Autoria própria, 2022.

Figura 14 – Lista das imagens presentes em um conjunto de dados



Fonte: Autoria própria, 2022.

4.2 Modelos

A aba *Modelos* é apresentada na Figura 15. Assim como a de *Dados*, é composta de uma única página. Nela são listados todos os modelos de arquitetura de rede neural que o DeepUAI tem disponível. Diferente da aba *Dados*, o usuário não é capaz de adicionar novos modelos à aplicação.

Dentro do componente visual de cada modelo há um botão com o título *Treinar*. Por meio deste botão o usuário pode iniciar o treinamento completo deste modelo em um conjunto de dados, criando uma nova aplicação. Os pesos iniciais, das aplicações geradas a partir dessa aba, são compostos por valores aleatórios e não tiram vantagem de nenhum treinamento prévio.

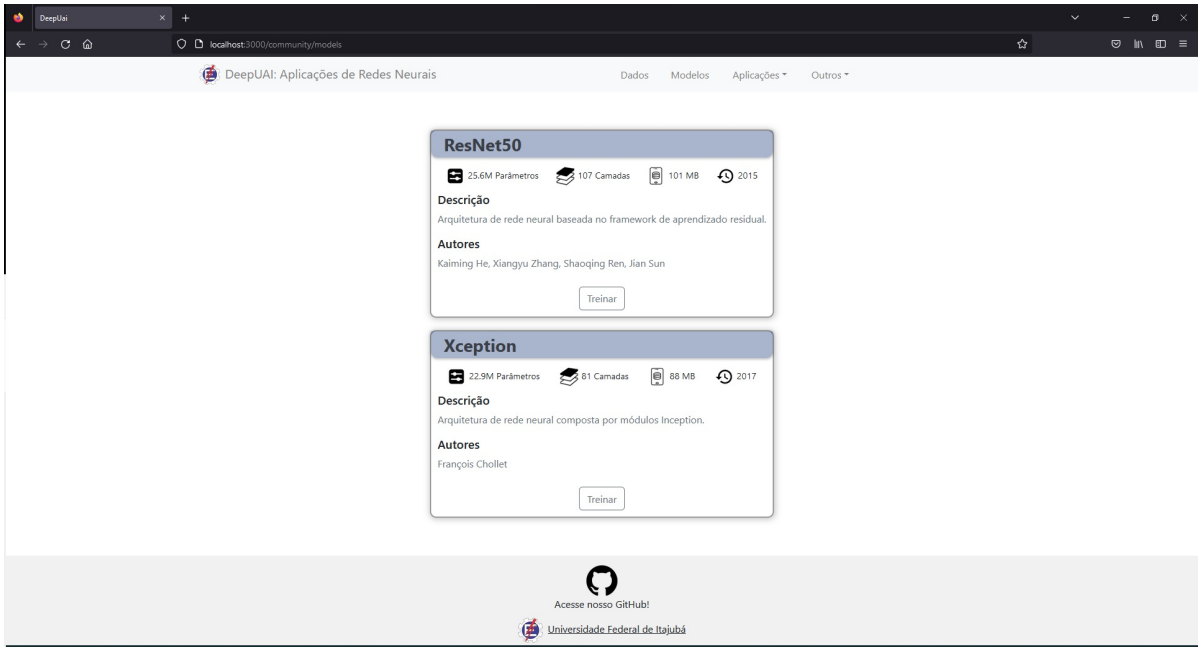
4.3 Aplicações

A aba *Aplicações* possui duas páginas: *Disponíveis* e *Na Fila*.

Em *Disponíveis* (Figura 16) estão listadas todas as aplicações de redes neurais disponíveis na plataforma, ou seja, modelos de arquiteturas de redes neurais que foram treinados com um conjunto de dados dentro da plataforma ou previamente carregados.

Assim como nos componentes visuais dos modelos, os componentes das aplicações possuem o botão *Treinar*, que possibilita que o usuário crie uma nova aplicação, escolhendo treiná-la em um novo conjunto de dados.

Figura 15 – Página com os modelos disponíveis na aplicação DeepUAI



Fonte: Autoria própria, 2022.

Não há diferença visual entre a funcionalidade de treinamento de modelos e a de treinamento de aplicações, apenas o fato de que uma aplicação, quando criada a partir de outra aplicação, faz uso do *transfer learning*, e não do treinamento completo da rede.

Nesta tela, também é possível visualizar o histórico de versões para determinada aplicação *deep learning*, uma vez que o processo de transferência de aprendizado pode ser realizado de forma recorrente. Assim, esse histórico é representado em forma de grafo como na Figura 17, logo após o usuário clicar sob a versão da aplicação, localizada ao lado direito de seu respectivo nome.

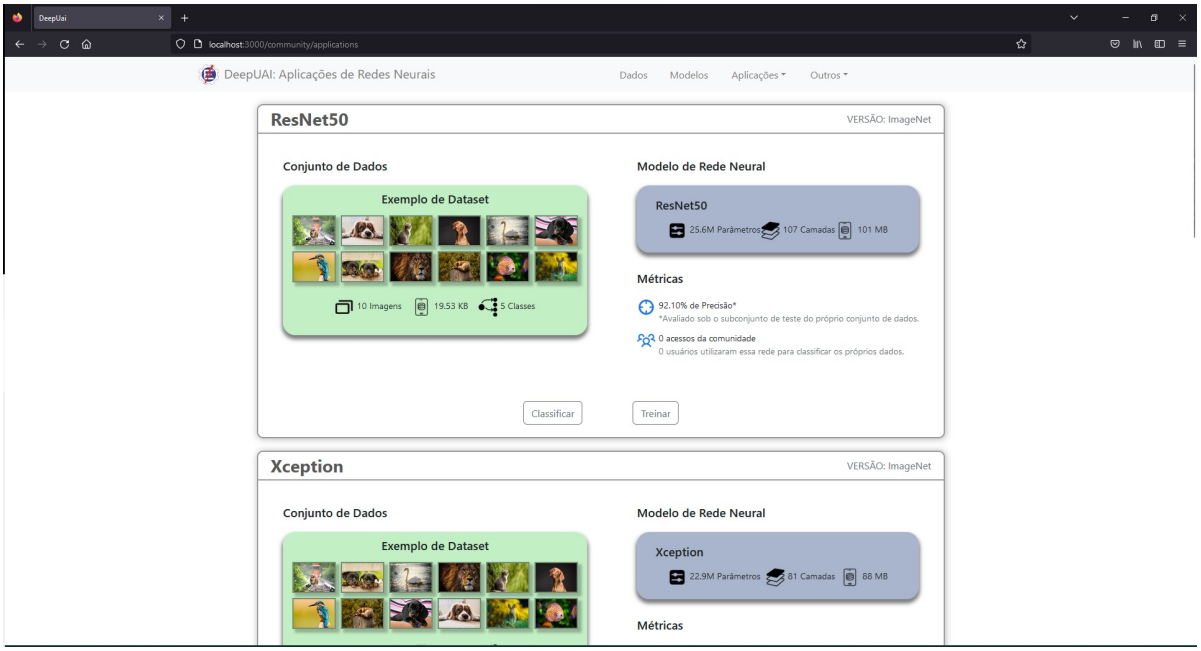
Além do botão *Treinar*, há o botão *Classificar*, que possibilita que o usuário envie uma imagem qualquer e receba, de forma gráfica, o resultado da classificação daquela imagem pela aplicação.

Já na página *Na Fila* (Figura 18) estão listadas as aplicações que já foram criadas pelos usuários, mas que ainda não completaram seu treinamento, sendo feita a diferenciação se o treinamento está sendo processado ou se está aguardando na fila.

4.4 Treinamento

Após a escolha do modelo ou aplicação *deep learning* desejada, a escolha do *dataset* pode ser realizada de duas maneiras diferentes: a partir do envio de um novo arquivo, no formato compactado (Figura 19), ou por um conjunto de dados já disponível no sistema (Figuras 20 e 21).

Figura 16 – Página com as redes neurais treinadas disponíveis na aplicação DeepUAI



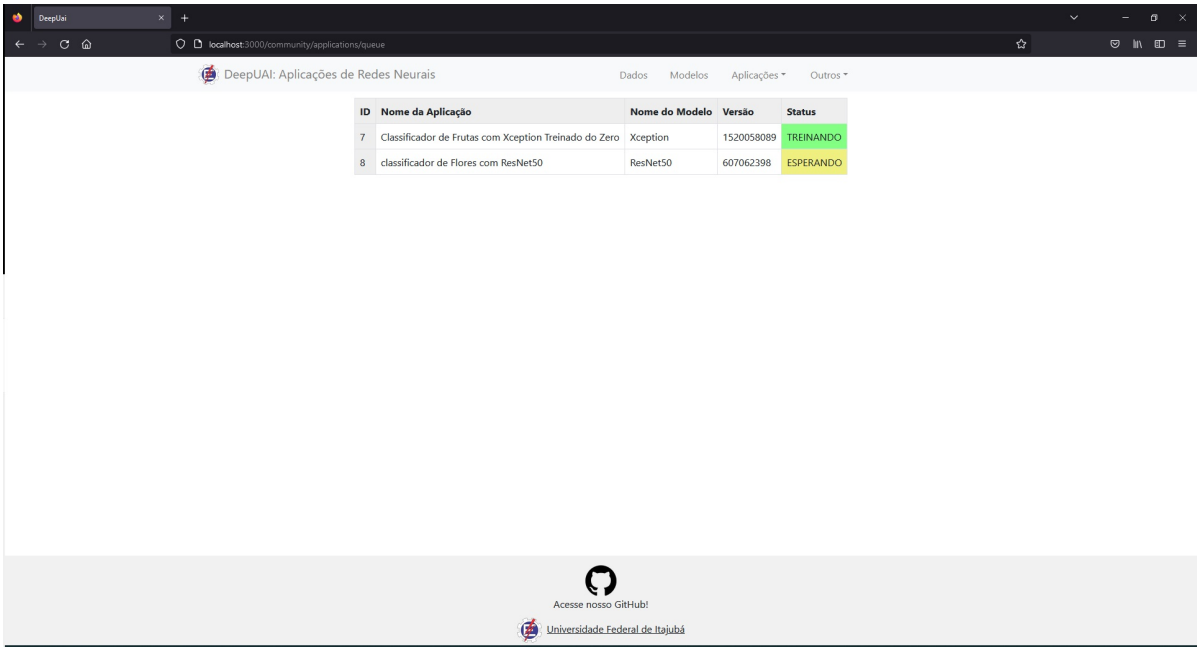
Fonte: Autoria própria, 2022.

Figura 17 – Página com o histórico de versões para uma aplicação presente na plataforma DeepUAI

Fonte: Autoria própria, 2022.

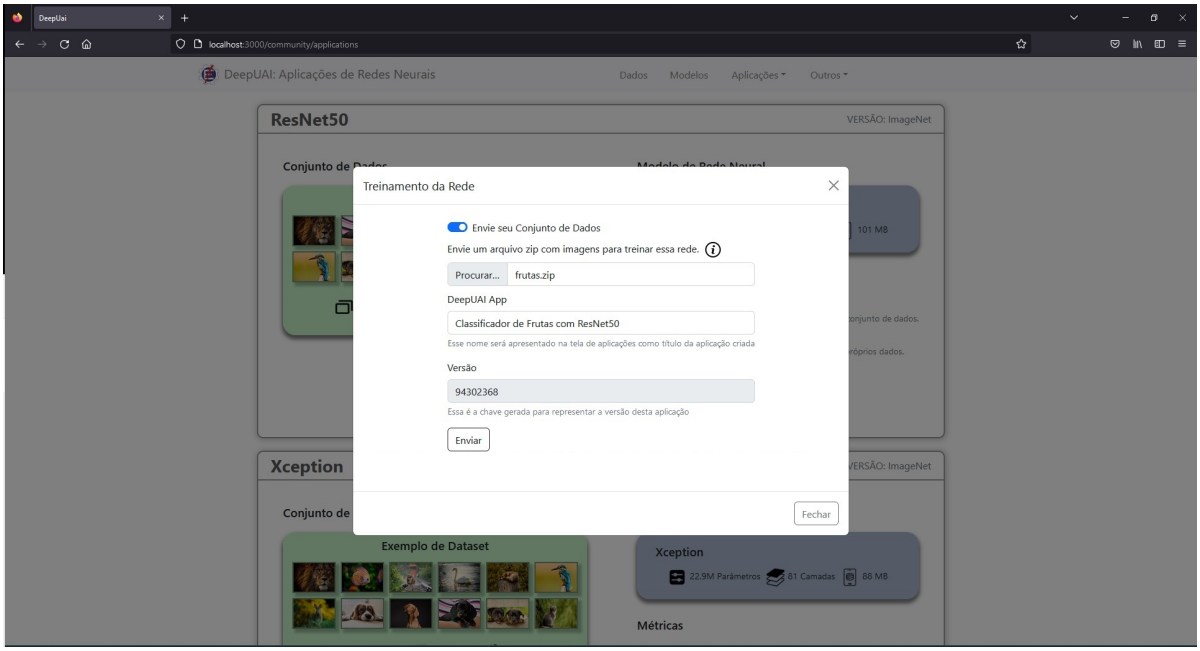
Ressalta-se ainda que, quando uma rede neural é treinada a partir de um arquivo compactado de imagens, esse conjunto de dados passa a fazer parte da tela de *datasets* disponíveis (Figura 13).

Figura 18 – Página com as redes neurais artificiais na fila para treinamento



Fonte: Autoria própria, 2022.

Figura 19 – Página onde é possível realizar o treinamento de determinada rede ou modelo selecionado a partir de um conjunto de dados importado pelo usuário como arquivo compactado

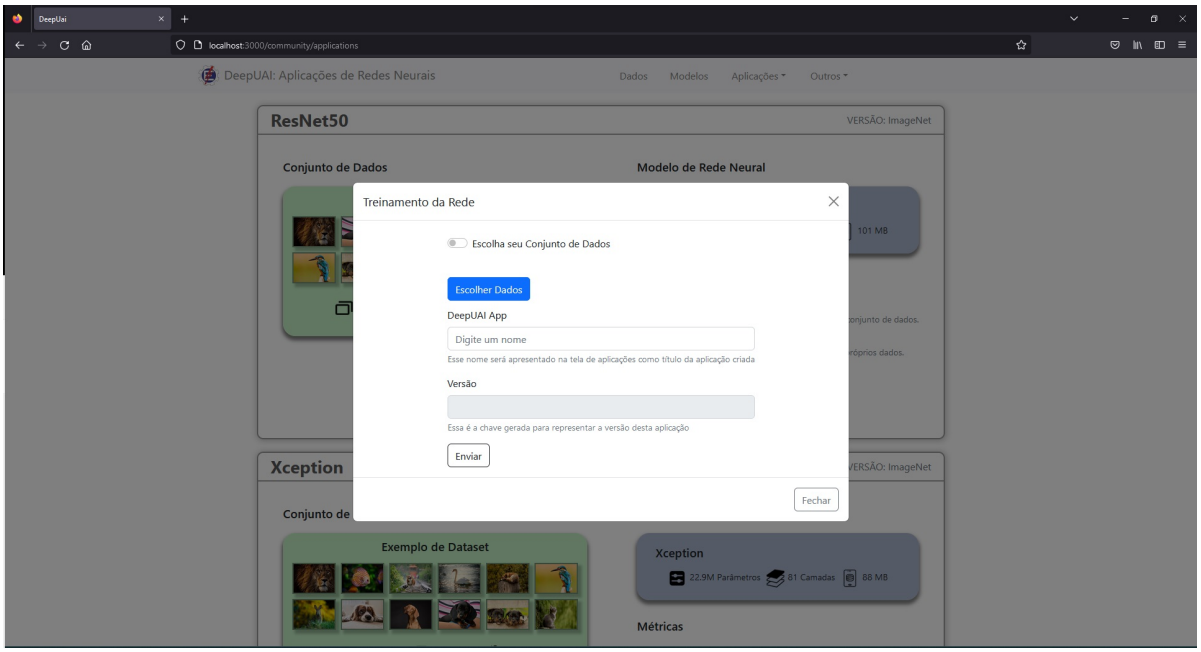


Fonte: Autoria própria, 2022.

4.5 Experimentos

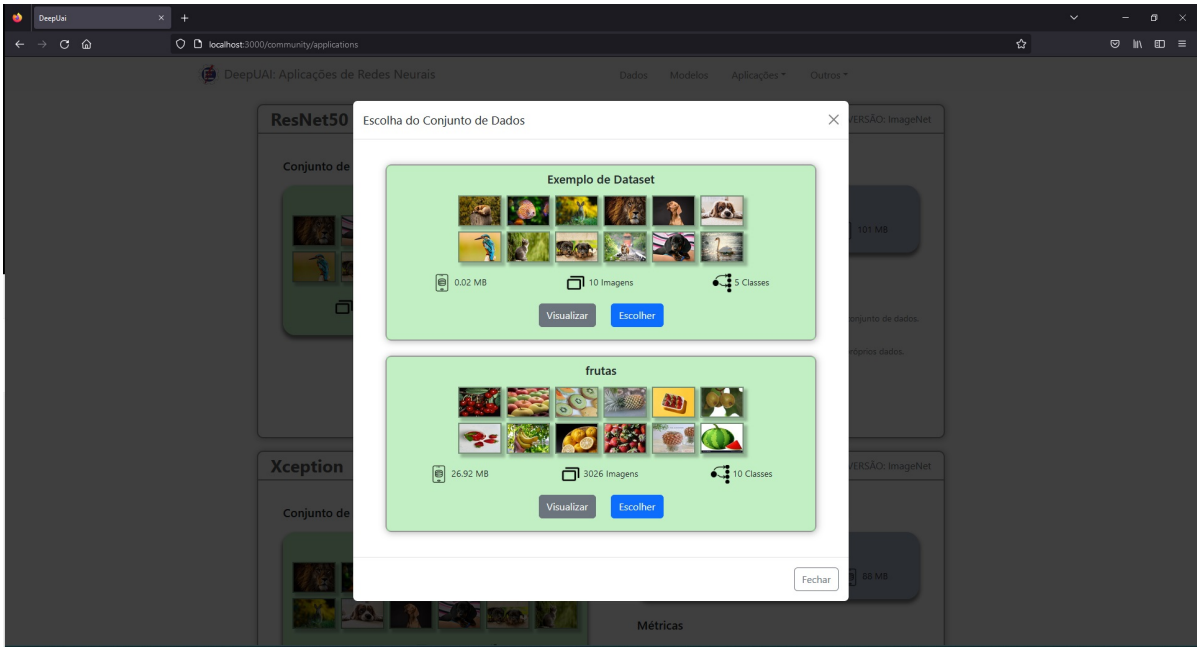
Para validar a aplicação *web*, foram realizados dois experimentos que consistiram em treinar aplicações de *deep learning* em um *dataset* e em seguida utilizá-las para classificar outras imagens similares às do *dataset*, tudo através do sistema DeepUAI.

Figura 20 – Página onde é possível realizar o treinamento de determinada rede ou modelo selecionado a partir de um conjunto de dados já disponível no sistema DeepUAI



Fonte: Autoria própria, 2022.

Figura 21 – Página onde é possível selecionar o dataset desejado dos disponíveis para realizar o treinamento



Fonte: Autoria própria, 2022.

4.5.1 Classificação de Frutas

Foi utilizado o modelo de arquitetura ResNet50, pré-treinada no *dataset* Imagenet, e um conjunto de dados - disponível de forma gratuita na plataforma Kaggle[1] - de

[1] Disponível em <https://www.kaggle.com/datasets/karimabdulnabi/fruit-classification10-class>

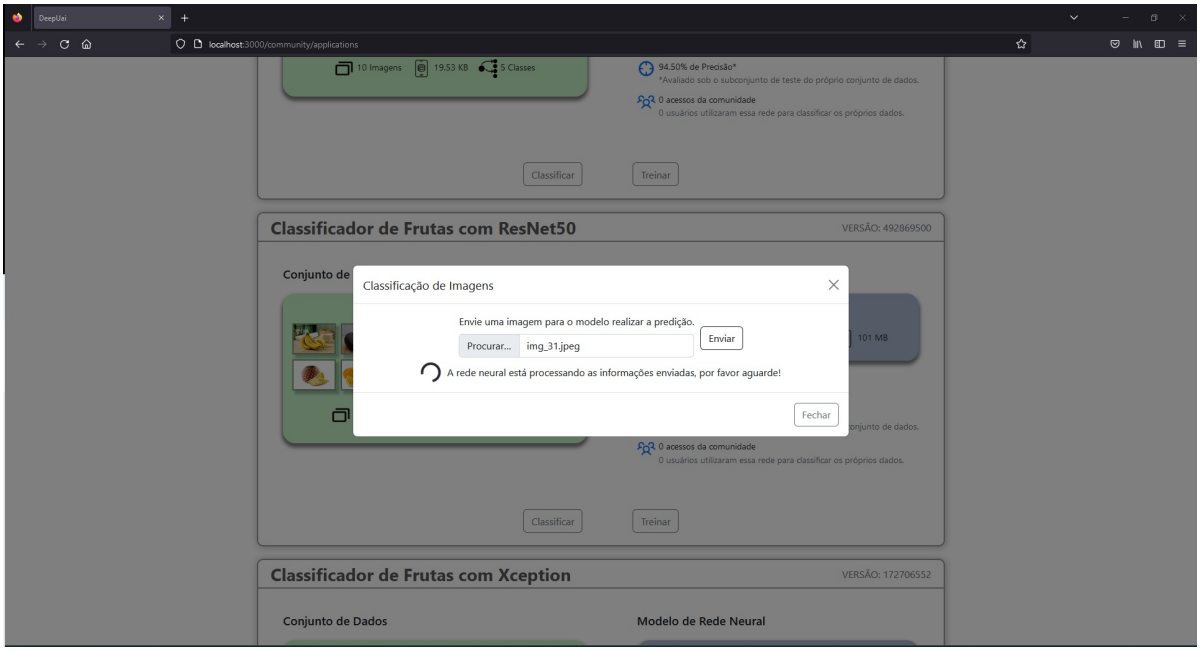
classificação de frutas, que é composto por 10 classes: maçã; kiwi; banana; cereja; laranja; manga; abacate; abacaxi e morango.

Após o treinamento foi feito o envio da imagem de um abacate (Figura 22), nomeado como "img_31.jpeg", para classificação (Figura 23). Foi obtido um resultado 100% de acerto. A Figura 24 demonstra o gráfico com 100% de acerto na classe de abacate e outras duas classes que a rede mostra como possibilidade, apesar do resultado ser certeiro. Essa taxa demonstra confiabilidade no funcionamento da aplicação, tendo em vista que a imagem utilizada para teste não estava presente no subconjunto de treinamento.

Figura 22 – Imagem utilizada para a funcionalidade de classificação na rede treinada com o *dataset* de frutas

Fonte: Parte do subconjunto de testes do *dataset* gratuito de frutas mencionado e disponível na plataforma Kaggle, 2022.

Figura 23 – Página onde é possível selecionar uma imagem e fazer sua classificação na rede neural selecionada

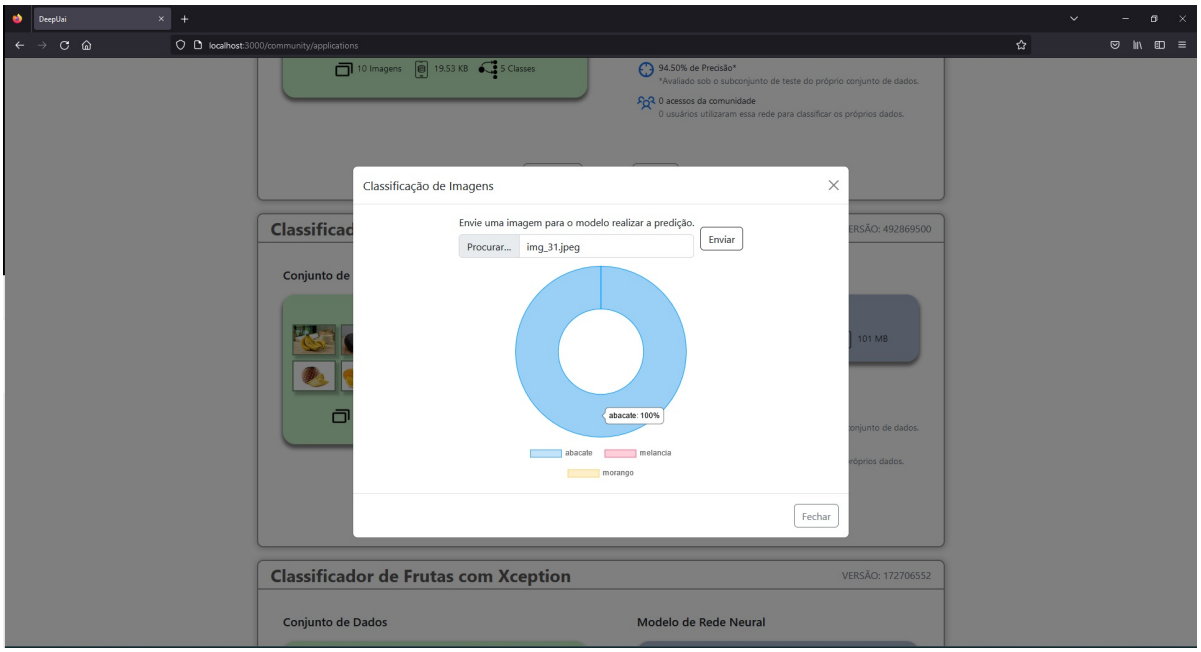


Fonte: Autoria própria, 2022.

4.5.2 Classificação de Sinais - *American Sign Language* (ASL)

Utilizando outra rede neural, representada na Figura 25, que também é definida pela arquitetura ResNet50 pré-treinada no *dataset* ImageNet, mas agora treinada com

Figura 24 – Página com a classificação, da imagem do abacate, após o processamento pelo servidor



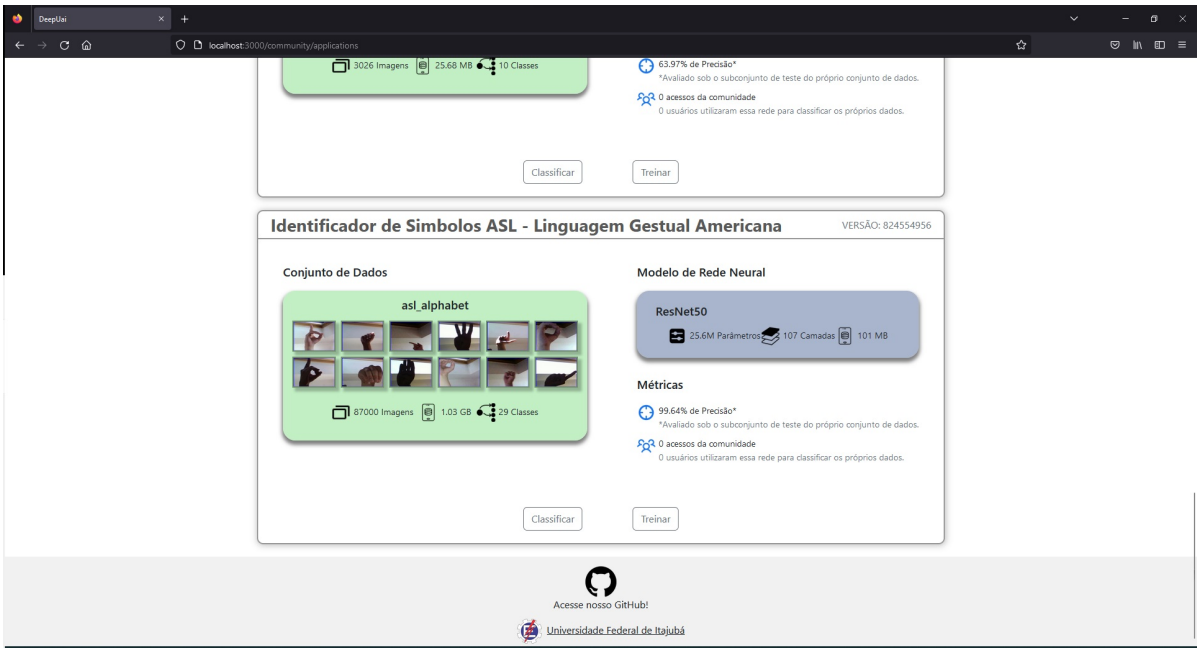
Fonte: Autoria própria, 2022.

um conjunto de imagens composto pelos símbolos da *American Sign Language* (ASL) - disponível de forma gratuita na plataforma Kaggle[2] e, em parte, representado na Figura 26 - realizou-se a classificação de uma imagem onde está presente o símbolo Y (Figura 27). O retorno da classificação também foi 100% de acerto, demostrando a confiabilidade do resultado, uma vez que a imagem classificada não faz parte do subconjunto de treinamento. O resultado demonstrado na interface pode ser visto no gráfico da Figura 28, onde identifica corretamente a letra Y e outras duas letras que poderiam ter alguma semelhança.

Realizou-se também a classificação, nesta mesma aplicação, de fotografias capturadas pelos autores a partir da *webcam* do *laptop*, capturando os seguintes símbolos: U, N, I, F, E e I (Figura 29). Após o processamento das imagens pela rede neural, obteve-se como resultado as respectivas classificações para cada carácter: 68,01% de chance de ser a letra B; 100,00% de chance de ser a letra N; 66,16% de chance de ser a letra X; 100,00% de chance de ser a letra F; 68,66% de chance de ser a letra E; e por fim, 99,99% de chance de ser a letra I. Os resultados também são apresentados na Tabela 3, contendo a letra desejada, letra classificada pela rede e sua porcentagem de certeza em colunas. Os gráficos da Figura 30 também demonstram essa mesma porcentagem para cada letra. É possível analisar através dessas informações que a rede neural acertou quatro dos seis símbolos classificados, obtendo uma taxa de acerto de 66,67% para esse exemplo. Porém, alguns fatores podem ter influenciado nestas classificações, como a qualidade das fotografias e o fato de determinados símbolos poderem não ter sido corretamente representados pelos

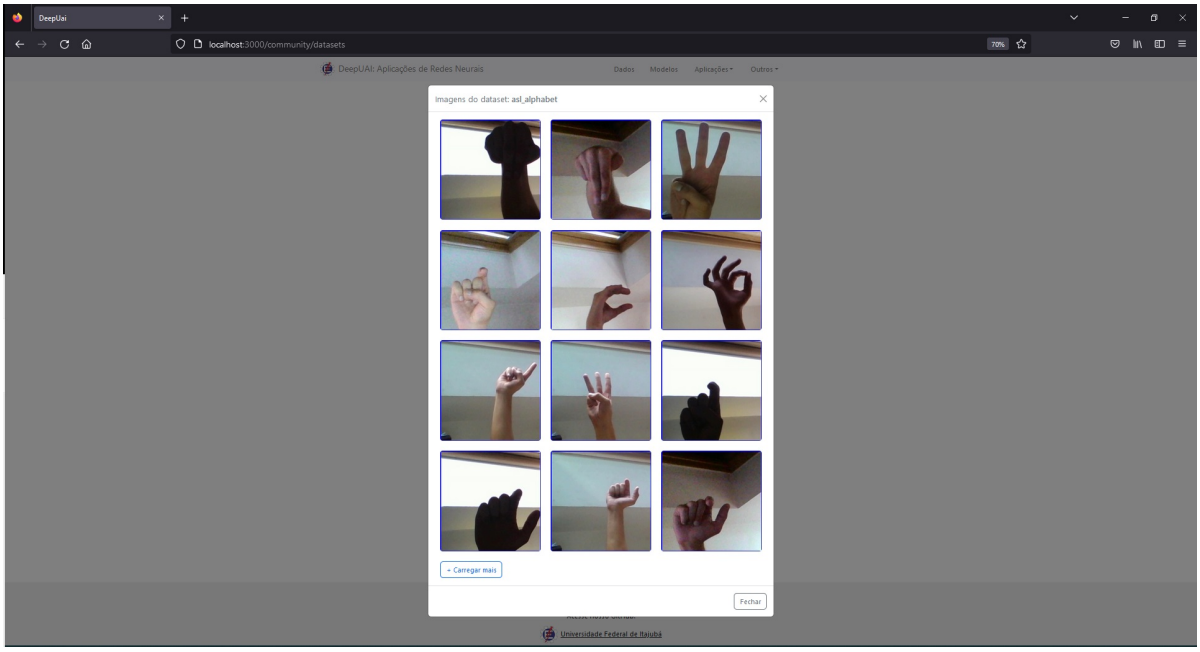
[2] Disponível em <https://www.kaggle.com/datasets/grassknoted/asl-alphabet>

Figura 25 – Rede neural treinada utilizando *transfer learning* e o conjunto de símbolos da Linguagem de Sinais Americana



Fonte: Autoria própria, 2022.

Figura 26 – Visualização de parte do *dataset* após sua disponibilização pela aplicação



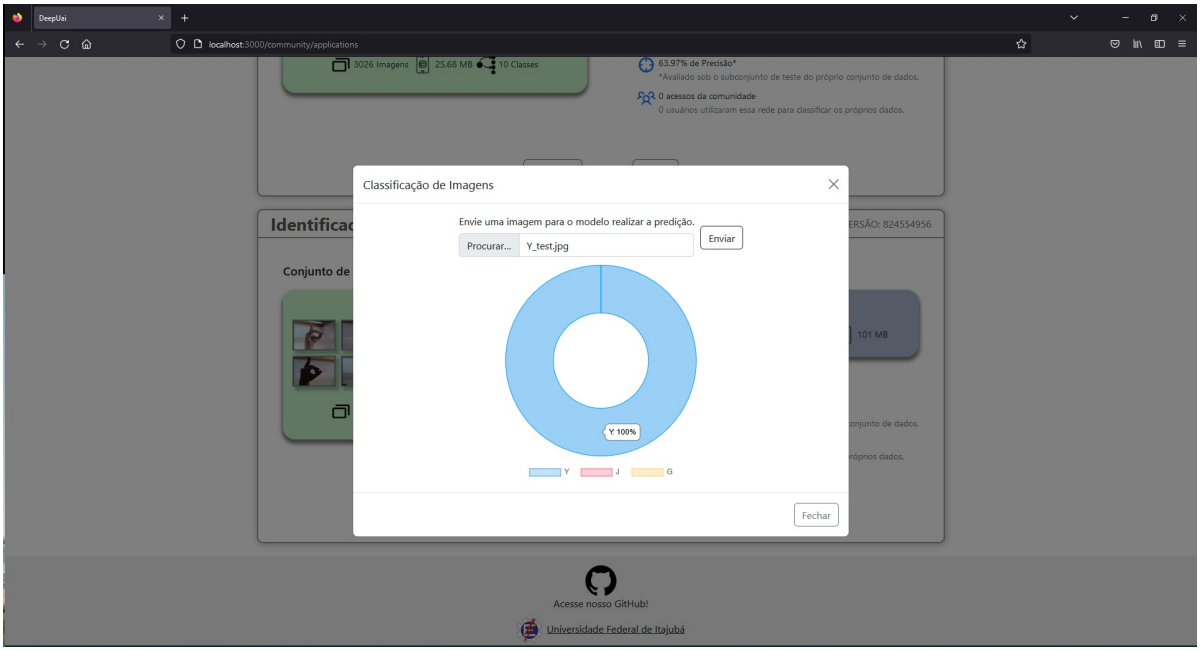
Fonte: Autoria própria, 2022.

autores, que não possuem fluência nessa linguagem e buscaram construir os sinais apenas a título de exemplo a partir da observação de imagens dos respectivos símbolos.

Figura 27 – Símbolo, que representa a letra Y, utilizado para a funcionalidade de classificação na rede neural

Fonte: Parte do subconjunto de testes do *dataset* gratuito de símbolos ASL mencionado e disponível na plataforma Kaggle, 2022.

Figura 28 – Classificação de uma imagem, a qual contém o símbolo Y, na rede neural treinada utilizando *transfer learning* e o *dataset* de símbolos ASL



Fonte: Autoria própria, 2022.

Tabela 3 – Previsões do modelo treinado para identificar símbolos ASL, utilizando fotografias tiradas a partir da *webcam* do *laptop*

Rótulo real da imagem	Classificação pela rede neural	Porcentagem de certeza (%)
U	B	68,01
N	N	100,00
I	X	66,16
F	F	100,00
E	E	68,66
I	I	99,99

Fonte: Autoria própria, 2022.

Figura 29 – Fotografias, referentes aos símbolos classificados, capturadas utilizando a *webcam* do *laptop*

(a) Letra U

(b) Letra N

(c) Letra I - 1

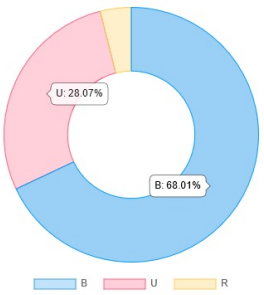
(d) Letra F

(e) Letra E

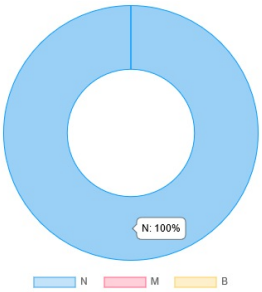
(f) Letra I - 2

Fonte: Autoria própria, 2022.

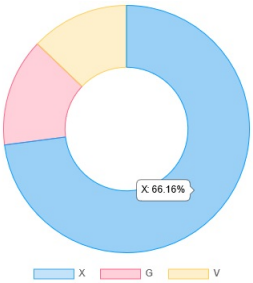
Figura 30 – Classificações pela rede tendo em vista os respectivos símbolos



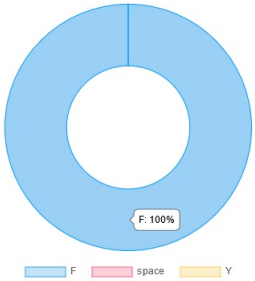
(a) Letra U



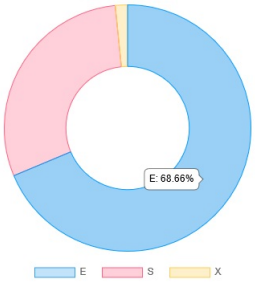
(b) Letra N



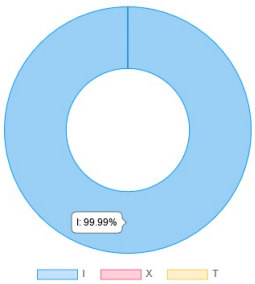
(c) Letra I - 1



(d) Letra F



(e) Letra E



(f) Letra I - 2

Fonte: Autoria própria, 2022.

5 Conclusão e Trabalhos Futuros

O objetivo deste trabalho foi construir uma interface gráfica *web* para permitir que pessoas sem conhecimento em programação possam criar suas próprias aplicações de classificação de imagens para validar e treinar modelos de redes neurais.

Os resultados obtidos neste trabalho mostram que é possível treinar uma nova aplicação de *deep learning* utilizando apenas a interface DeepUAI e obter resultados consistentes ao utilizá-la posteriormente na classificação de novas imagens.

Ainda assim, os resultados não foram livres de erros. Na definição da etapa de treinamento, muitos dos hiper parâmetros foram escolhidos de forma arbitrária, sem uma análise mais profunda do problema a se resolver, o que pode levar a redes com baixa precisão.

Buscando resumir para o usuário o processo de construção de uma nova aplicação em apenas dois passos, escolha dos dados e escolha do modelo ou aplicação base, foi necessário abrir mão do controle de algumas configurações da rede que desenvolvedores normalmente têm acesso ao utilizar uma API como a Keras.

Portanto, conclui-se que, abrindo mão de algumas configurações, é possível recriar, no formato de uma aplicação *web*, o processo de treinamento de uma rede de *deep learning* para classificação de imagens com arquitetura pré-definida, como também o processo de classificação de novas imagens utilizando essa rede treinada.

As principais dificuldades enfrentadas no desenvolvimento estavam presentes na construção da interface gráfica e sua conexão com as APIs. Transformar uma API complexa como a Keras em uma interface gráfica envolve entender a ferramenta, suas funcionalidades, e, principalmente, uma forma de disponibilizar seus recursos através de uma plataforma visual. O processo de *design* da interface para pessoas de diversos níveis de conhecimento também foi um desafio, dado que a ideia deste sistema foi ser de fácil uso, porém a curva de aprendizado varia para cada indivíduo.

Por fim, alguns pontos foram levantados para trabalhos futuros: o aprimoramento da página de aplicações na fila, a fim de mostrar determinados registros do andamento do processamento pelo servidor, como tempo estimado e informações da *epoch* sendo treinada no momento; possibilitar que o usuário classifique uma imagem em mais de uma aplicação de forma simultânea, trazendo a comparação entre as predições retornadas por diferentes redes neurais; disponibilizar uma quantidade maior de arquiteturas conhecidas e presentes na literatura; e possibilitar com que usuários criem suas próprias arquiteturas, escolhendo o tipo e a quantidade de camadas desejadas, bem como outros parâmetros da rede neural.

Referências

AGRAWAL, P. et al. Data platform for machine learning. In: *Proceedings of the 2019 International Conference on Management of Data*. [S.l.: s.n.], 2019. p. 1803–1816. Citado na página 35.

ALBAWI, S.; MOHAMMED, T. A.; ALZAWI, S. Understanding of a convolutional neural network. In: . [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 24 e 25.

BARBOSA, X. de C. Breve introdução à história da inteligência artificial. *Jamaxi*, v. 4, n. 1, 2020. Citado na página 17.

BHALLA, A.; GARG, S.; SINGH, P. Present day web-development using reactjs. *International Research Journal of Engineering and Technology*, v. 7, 5 2020. Citado 2 vezes nas páginas 35 e 36.

BHARDWAJ, A. et al. Datahub: Collaborative data science & dataset version management at scale. *arXiv preprint arXiv:1409.0798*, 2014. Citado na página 33.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. Citado na página 30.

CHOLLET, F. *Deep Learning With Python*. 20 Baldwin Road, PO Box 761, Shelter Island, NY 11964: Manning Publications Co., 2018. Citado na página 32.

CHTips. *Advantages and Disadvantages of GUI | Benefits and Drawbacks of GUI*. 2022. Disponível em: <https://www.chtips.com/computer-fundamentals/advantages-and-disadvantages-of-gui/>. Citado na página 19.

DVC. *DVC Documentation*. 2022. Disponível em: <https://dvc.org/doc>. Citado na página 36.

Edge Impulse. *About*. 2022. Disponível em: <https://www.edgeimpulse.com/about>. Citado na página 37.

Edge Impulse. *Environment Conservation*. 2022. Disponível em: <https://www.edgeimpulse.com/solutions/environment>. Citado na página 37.

FINLEY, T. K. The democratization of artificial intelligence: One library's approach. *Information Technology and Libraries*, v. 38, n. 1, p. 8–13, 2019. Citado na página 18.

Furtunate Business Insights. *Machine Learning (ML) Market Size, Share & COVID-19 Impact Analysis, By Component (Solution, and Services), By Enterprise Size (SMEs, and Large Enterprises), By Deployment (Cloud and On-premise), By End-user (Healthcare, Retail, IT and Telecommunication, BFSI, Automotive and Transportation, Advertising and Media, Manufacturing, and Others), and Regional Forecast, 2022-2029*. 2022. Disponível em: <https://www.fortunebusinessinsights.com/machine-learning-market-102226>. Citado na página 19.

Google Cloud. *Guia de ML inclusivo*. 2022. Disponível em: <https://cloud.google.com/inclusive-ml/>. Citado na página 38.

HAGHIGHAT, E.; JUANES, R. Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 373, p. 113552, 2021. Citado na página 36.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>. Citado na página 29.

KRISHNA, S. T.; KALLURI, H. k. Deep learning and transfer learning approaches for image classification. 06 2019. Citado na página 33.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, v. 25, 01 2012. Citado 2 vezes nas páginas 27 e 28.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na página 23.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, p. 2278 – 2324, 12 1998. Citado 2 vezes nas páginas 26 e 27.

MARQUES, L. S.; WANGENHEIM, C. Gresse von; HAUCK, J. C. Teaching machine learning in school: A systematic mapping of the state of the art. *Informatics in Education*, Vilnius University Institute of Data Science and Digital Technologies, v. 19, n. 2, p. 283–321, 2020. Citado na página 23.

MIAO, H. et al. Modelhub: Deep learning lifecycle management. In: IEEE. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. [S.l.], 2017. p. 1393–1394. Citado na página 33.

MITIĆ, V. et al. Benefits of artificial intelligence and machine learning in marketing. In: SINGIDUNUM UNIVERSITY. *Sinteza 2019-International scientific conference on information technology and data related research*. [S.l.], 2019. p. 472–477. Citado na página 17.

MONDAL, B. Artificial intelligence: state of the art. *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, Springer, p. 389–425, 2020. Citado na página 23.

Neurocle. *Core Technology Neurocle AutoDL*. 2022. Disponível em: <https://www.neuro-cle.com/en/core-technology>. Citado na página 38.

PATEL, S. A comprehensive analysis of convolutional neural network models. *International Journal of Advanced Science and Technology*, v. 29, p. 771–777, 01 2020. Citado 2 vezes nas páginas 23 e 24.

QIN, Z. et al. How convolutional neural network see the world - A survey of convolutional neural network visualization methods. *CoRR*, abs/1804.11191, 2018. Disponível em: <http://arxiv.org/abs/1804.11191>. Citado na página 24.

RAWAT, P.; MAHAJAN, A. N. Reactjs: A modern web development framework. *International Journal of Innovative Science and Research Technology*, v. 5, 11 2020. Citado 2 vezes nas páginas 35 e 36.

ROUHIAINEN, L. Inteligencia artificial. *Madrid: Alienta Editorial*, 2018. Citado na página 17.

SIKPA, D. et al. Automated detection and quantification of breast cancer brain metastases in an animal model using democratized machine learning tools. *Scientific reports*, Nature Publishing Group, v. 9, n. 1, p. 1–8, 2019. Citado na página 18.

SIMONYAN, K.; ZISSERMAN, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1409.1556>. Citado na página 28.

SZEGEDY, C. et al. Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. Citado 2 vezes nas páginas 29 e 30.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1905.11946>. Citado na página 30.

THE ROYAL SOCIETY. *Machine learning: the power and promise of computers that learn by example*. 2017. Citado na página 23.

WURZEL, P.; MARTINS, M. O. Classificação de imagens de mamografia com machine learning no auxílio de diagnósticos de câncer de mama. *Disciplinarum Scientia| Naturais e Tecnológicas*, v. 23, n. 2, p. 1–17, 2022. Citado na página 18.

ZEILER, M. D.; FERGUS, R. *Visualizing and Understanding Convolutional Networks*. arXiv, 2013. Disponível em: <https://arxiv.org/abs/1311.2901>. Citado 2 vezes nas páginas 27 e 28.

ZHANG, Y. et al. Datalab: a version data management and analytics system. In: *Proceedings of the 2nd International Workshop on BIG Data Software Engineering*. [S.l.: s.n.], 2016. p. 12–18. Citado na página 34.

Apêndices

APÊNDICE A – Detalhes de Implementação

Todo código desenvolvido pode ser consultado através de seus respectivos repositórios no GitHub[1]. O desenvolvimento do *front-end* seguiu a seguinte organização de diretórios da Figura 31. Na pasta *public* estão localizados os arquivos estáticos, como ícones e imagens, além da página index.html[2]. No diretório *src*, e em especifico no subdiretório *components*, encontram-se todos os componentes React construídos para representar as páginas do sistema; a estruturação do modelo de interface utilizando cabeçalho, conteúdo e rodapé; e os componentes específicos para as funções da aplicação gráfica proposta. Ressalta-se ainda a presença dos arquivos index.js[3] e Routes.js[4] na pasta *src*.

Já o servidor é estruturado segundo a Figura 32. Na pasta *assets* são armazenadas as redes neurais convolucionais disponíveis na aplicação, bem como os conjuntos de dados importados para o sistema ou utilizados para realizar o treinamento de determinada rede. No diretório *applications* está localizado a classe responsável por abstrair uma aplicação *deep learning*, isto é, uma rede neural convolucional qualquer presente no *software* DeepUAI. Em *controllers* encontra-se o código que realiza a interface entre uma requisição de treinamento e a geração de uma nova mensagem na fila do RabbitMQ, além do código responsável pela criação de um novo conjunto de imagens. Na pasta *db*, as configurações de conexão e gerenciamento do banco de dados. No diretório *rabbitmq*, a configuração do produtor e consumidor relacionados ao processo de treinamento. Em *routers*, os endereços disponíveis para serem acessados através de um cliente HTTP. E no diretório *utils*, os procedimentos utilizados para treinamento, classificação, preprocessamento e manipulação de arquivos. Ademais, é valido a explicação da presença do arquivo docker-compose.yml na pasta *src*, o qual tem a finalidade de inicializar o container relacionado ao *software* RabbitMQ.

Por fim, a API é organizada como mostrado na Figura 33. No diretório *config* estão presentes o arquivo com as rotas disponíveis para acessar o serviço REST, os *middlewares* necessários para o funcionamento da API como desejado, e a configuração de conexão com o banco de dados utilizando-se da biblioteca Knex. A pasta *migrations* possui as tabelas à serem criadas no banco. E no diretório *src*, encontra-se o código relacionado às validações durante determinado *request*, bem como os controladores para cada uma das rotas presentes na API. Para este caso, o arquivo docker-compose.yml é o responsável por inicializar o banco de dados PostgreSQL.

---

1 Repositórios do projeto disponíveis em <https://github.com/deepuai>

2 *DocumentoHyperText Markup Language* (HTML) que define o elemento *div root*, o qual é utilizado como ponto de entrada para a aplicação React desenvolvida.

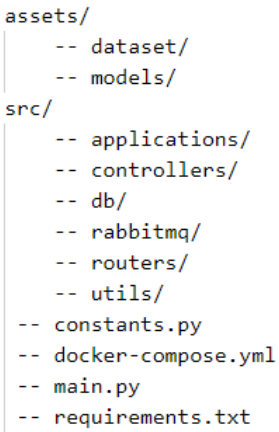
3 Código JavaScript responsável por capturar a *div root* presente no index.html e fazer com que ela receba o valor do componente App.js - o qual representa a aplicação.

4 Arquivo com as definições de rotas disponíveis para serem acessadas pelos usuários.

Figura 31 – Organização de diretórios da interface gráfica

Fonte: Autoria própria, 2022.

Figura 32 – Organização de diretórios do servidor



Fonte: Autoria própria, 2022.

Figura 33 – Organização de diretórios da API

```
config/
migrations/
src/
    -- controllers/
    -- validations.js
-- docker-compose.yml
-- index.js
-- knexfile.js
```

Fonte: Autoria própria, 2022.