UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - UTFPR
DAINF - DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
DAELN - DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

CAROLINE PEREIRA DE SENA

ESPELHO INTELIGENTE PARA USO DOMÉSTICO CONTROLADO POR GESTOS E FALA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2020

CAROLINE PEREIRA DE SENA

ESPELHO INTELIGENTE PARA USO DOMÉSTICO CONTROLADO POR GESTOS E FALA

Proposta de Trabalho de Conclusão de Curso apresentada ao Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para a obtenção do título de Graduando.

Orientador: Daniel Rossato
DAELN - Departamento Acadêmico de Eletrônica -UTFPR

CURITIBA
2020

CAROLINE PEREIRA DE SENA

ESPELHO INTELIGENTE PARA USO DOMÉSTICO CONTROLADO POR GESTOS E FALA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 17 / novembro / 2020

_____

João Alberto Fabro
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Ronier Rohrich
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Daniel Rossato de Oliveira
Mestrado
Universidade Tecnológica Federal do Paraná

CURITIBA

2022

RESUMO

SENA, Caroline Pereira de. Espelho Inteligente para uso doméstico controlado por gestos e fala. 2020. 83 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná - UTFPR. Curitiba, 2020.

O rápido avanço tecnológico dos tempos atuais muda, também de maneira rápida, a forma como as pessoas interagem com o mundo a sua volta. A constante necessidade por informação de acesso rápido e fácil, juntamente com o avanço tecnológico, resultou no surgimento de interfaces tecnológicas inovadoras, chamadas de dispositivos inteligentes, associadas ao conceito de Internet das Coisas. Dispositivos inteligentes são capazes de se conectarem a outros dispositivos, com os quais trocam informações e permitem tornar a vida das pessoas mais fácil e conveniente. A praticidade que dispositivos inteligentes oferecem os tornou cada vez mais populares e presentes no dia a dia das pessoas, principalmente dentro de suas casas. Nesse contexto, este trabalho tem como objetivo realizar a modelagem e desenvolvimento de um *Smart Mirror*, um espelho inteligente que disponibiliza informações úteis ao usuário. O espelho é capaz de reconhecer o usuário e mostrar informações que podem ser personalizadas por meio de um aplicativo Android. Além disso, o usuário pode navegar pela interface do espelho através de comandos por gestos e fala.

Palavras-chave: *Smart Mirror*. Internet das coisas. *Deep Learning*. Visão Computacional.

ABSTRACT

SENA, Caroline Pereira de. Título em inglês. 2020. 83 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná - UTFPR. Curitiba, 2020.

The fast technological advance of the present times rapidly changes how people interact with the world around them. The constant need for fast and easy information access, along with technological advancement, has resulted in the emergence of innovative technological interfaces, called Smart Devices, associated with the concept of Internet of Things. Smart devices are capable of connecting with other devices, with which they exchange information to make people's lives easier and more convenient. The practicality that smart devices offer has made them increasingly popular and present in people's daily lives, especially inside their homes. In this context, this project aims to model and develop a Smart Mirror that provides useful information to the user. The mirror is able to recognize the user and show information that can be customized through an Android application. In addition, the user can navigate through the mirror interface through gestures and speech commands.
Keywords: Smart Mirror. Internet of Things. Deep Learning. Computer Vision.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface* - Interface de Programação de Aplicações.
ML	*Machine Learning* - Apredizagem de Máquina.
DL	*Deep Learning* - Aprendizagem Profunda.
CNN	*Convolutional Neural Network* - Rede Neural Convolucional.
IoT	*Internet of Things* - Internet das Coisas.
SBC	*Single Board Computer* - Computador de Placa Única.
HTTP	*Hipertext Transfer Protocol* - Protocolo de Transferência de Hipertexto.
PDI	Processamento Digital de Imagens.

SUMÁRIO

1 INTRODUÇÃO

Nas últimas duas décadas, o rápido avanço tecnológico mudou a forma como as pessoas se relacionam com o mundo ao seu redor e permitiu o surgimento de interfaces tecnológicas inovadoras. As pessoas começaram a passar mais tempo do seu dia utilizando seus *smartphones*, tanto para se comunicarem e obterem informações úteis de forma fácil, quanto para se entreterem. Isso as tornou cada vez mais dependentes desses dispositivos para tornar seus dias mais práticos e eficientes, e também ajudou no aumento pelo interesse em outros dispositivos que tivessem um objetivo parecido.

O desejo e necessidade por informação rápida e de acesso fácil, abriu espaço para o mercado de dispositivos *smart* (inteligentes). SILVERIO-FERNÁNDEZ, RENUKAPPA e SURESH (2018) define dispositivos *smart* como dispositivos eletrônicos cientes de seu próprio contexto, que conseguem se comunicar com outros e trocar dados, e que são capazes de realizar computação autônoma, ou seja, realizar tarefas de forma autônoma. Dispositivos inteligentes são o centro de um conceito que surgiu há pouco tempo: *Internet of Things* (IoT), em português, Internet das Coisas. Segundo BAHGA e MADISETTI (2014), IoT compreende a configuração, controle, processamento de dados e comunicação de dispositivos que tradicionalmente não estão associados à Internet, mas passam a ser conectados a ela, com o objetivo de executar aplicações importantes em prol de um objetivo do usuário ou máquina.

Dispositivos inteligentes, no contexto de IoT, podem se conectar a outros dispositivos e à internet, normalmente através de protocolos de comunicação sem fio, trocando e processando informações e usualmente permitindo que sejam controlados de maneira mais prática. Eles podem tornar a vida das pessoas mais fácil e conveniente, ajudar no dia a dia, na segurança de suas casas e em diversos aspectos da vida cotidiana. Alguns exemplos de dispositivos inteligentes são os auto-falantes de assistentes virtuais Alexa da Amazon[1] e Google Home, bem como lâmpadas que podem ser controladas por voz e aplicativos móveis, câmeras de segurança que reconhecem intrusos e termostatos inteligentes, entre muitos outros. Nos últimos anos, ficou bastante evidente o aumento no consumo e na variedade de dispositivos inteligentes disponíveis no mercado.

No nicho de dispositivos inteligentes surgiram também os *Smart Mirrors*, espelhos inteligentes, que consistem em espelhos com uma interface integrada, que disponibilizam diversas informações que podem ser úteis no dia a dia do usuário, como por exemplo, hora, previsão meteorológica, informações de trânsito, notícias, entre outros, de maneira simples e esteticamente agradável. Tais dispositivos obtém informações através de *application programming interfaces* (APIs), bancos de dados e outros dispositivos ou aplicativos para dispositivos móveis aos quais possam se conectar. Muitos deles também oferecem funções

[1] <https://developer.amazon.com/pt-BR/alexa> (acessado em 26/06/2020)

mais complexas que envolvem o uso de sensores.

Atualmente, existem alguns tipos de *Smart Mirror* disponíveis comercialmente. Eles podem ser separados em três principais áreas de aplicação: detecção, acompanhamento e exibição de informações relacionadas à saúde do usuário; auxílio na área estética, com opções que ajudam a simular o uso de maquiagens e roupas, normalmente com o uso de realidade aumentada; e o mais comum, uso doméstico, com exibição de informações úteis no cotidiano.

Alguns exemplos de *Smart Mirrors* comerciais são o CareOS [2], o Mango Mirror [3], o Embrace Smart Mirror [4], o Evervue Smart Mirror [5] e o My Smart Mirror [6]. O CareOS é uma plataforma aberta voltada para cuidados com a saúde e estética, possuindo funções como análise de pele, testes de visão e tutoriais de realidade aumentada para maquiagem. O Mango Mirror conecta-se a outros dispositivos inteligentes e *wearable* [7], e mostra informações sobre a saúde e nutrição do usuário, bem como notícias, calendário e tempo, entre outras coisas. O My Smart Mirror usa integração com Alexa para ser controlado por voz e disponibiliza informações de transporte público e trânsito de Londres. O Embrace Smart Mirror possui controle por voz e toque, é compatível com sistemas de automação domésticos e permite controlar sensores e luzes, entre outras funções. Esses exemplos mostram a diversidade de usos e aplicações de espelhos inteligentes em ambiente doméstico.

É possível também encontrar *online* diversos exemplos de projetos não comerciais de *Smart Mirror*, muitos deles baseados em uma plataforma de software open-source chamada MagicMirror[8], cuja proposta é oferecer uma interface modular para espelhos inteligentes facilmente customizável, com diversos módulos prontos para disponibilização de informações e integração com outros dispositivos inteligentes, além de aplicativos já existentes para permitir sincronização de dados do usuário.

Como exemplos de projetos com artigos podem ser citados: *FitMirror* (BESSERER et al., 2016), que se propõe a oferecer meios de incentivar o usuário a buscar um estilo de vida mais saudável; e o projeto de RAHMAN et al. (2010) que se propõe a utilizar Realidade Aumentada e Renderização 3D para auxiliar o usuário na escolha de produtos de maquiagem. Dentro desse contexto, este trabalho se propõe a desenvolver um espelho inteligente integralmente. O *Smart Mirror* proposto deve ser capaz de identificar o usuário que se coloque na sua frente e mostrar informações personalizadas por esse usuário. A personalização poderá ser feita por meio de um aplicativo móvel instalado no *smartphone*

[2] <https://www.care-os.com/> (acessado em 28/06/2020)
[3] <https://www.mangomirror.com/> (acessado em 28/06/2020)
[4] <https://www.embracesmartmirror.com/> (acessado em 28/06/2020)
[5] <https://www.evervuetv.com/smartmirror/> (acessado em 28/06/2020)
[6] <https://www.mysmartmirror.co.uk/> (acessado em 28/06/2020)
[7] Dispositivos utilizados no corpo. Exemplos comuns são relógios que exibem informações relacionadas à saúde, como frequência de batimentos cardíacos, número de passos dados pelo usuário, entre outros
[8] <https://magicmirror.builders/> (acessado em 28/06/2020)

do usuário, que permite que ele indique sua cidade, crie uma lista de tarefas e também uma lista de eventos. O aplicativo também deve permitir que o usuário omita alguma informação na interface, caso não tenha interesse em visualizá-la. O espelho deve poder ser controlado através de comandos de voz e gestos específicos de uma das mãos.

Os projetos citados anteriormente utilizam voz ou toque no espelho (com o uso de sensores) como forma de interação com a interface; assim, em comparação, o controle por gestos (sem toque) é uma alternativa interessante, por permitir o controle silencioso e sem contato com o espelho.

Para a implementação do sistema proposto, algumas áreas do conhecimento possuem um papel fundamental no desenvolvimento do projeto, sendo descritas de forma breve a seguir:

- **Inteligência Artificial**: é um ramo da ciência da computação que busca construir sistemas que consigam simular a inteligência e capacidade de aprendizado humanos e que, dessa forma, consigam tomar decisões por conta própria; utilizando para isso técnicas que buscam encontrar padrões em conjuntos de dados.
- **Internet das Coisas**: compreende a relação entre todo tipo de dispositivo que se conecta à Internet e a outros dispositivos, os protocolos de comunicação utilizados e os dados processados por eles, bem como a troca dinâmica de informações.
- **Banco de Dados**: são dados armazenados digitalmente de forma organizada, de forma a facilitar sua utilização, atualização e remoção por aplicações.
- **Comunicação *Wireless***: ramo de Telecomunicações que consiste em técnicas para transferência de dados entre máquinas, dispositivos e sistemas sem o uso de cabos.
- ***Single-board Computers***: computadores de placa única, em português, são computadores construídos em uma única placa de circuito, possuindo microprocessador, memória, interfaces de entrada e saída, entre outros componentes necessários em um computador. Por serem compactos, normalmente eles possuem capacidade de processamento computacional bastante limitada, mas custo reduzido.
- **Application Programming Interface (API)**: são interfaces criadas em aplicações que permitem que elas sejam programaticamente acessadas por outras aplicações, sem que seja necessário saber como elas foram implementadas.

1.1 OBJETIVOS

O objetivo geral desse projeto é desenvolver um *Smart Mirror*, ou Espelho Inteligente, que consiste em uma interface contendo informações personalizáveis pelo usuário, exibida em um monitor anexado a uma estrutura com um espelho. A interface deve ser personalizável através de um aplicativo *mobile* e passível de interação através de voz e gestos.

1.1.1 OBJETIVOS ESPECÍFICOS

Além do objetivo geral, este projeto tem como objetivos específicos os seguintes pontos:

1. Fazer a revisão teórica de conceitos de banco de dados, API, *deep learning* e visão computacional, redes de computadores e protocolos de comunicação *wireless*, bem como revisão de assuntos relacionados;
2. Modelager e desenvolver um *Smart Mirror* personalizável e interagível, com a descrição da implementação de cada parte e dos problemas e dificuldades encontrados durante o desenvolvimento, bem como os critérios para validar o funcionamento do sistema como um todo;
3. Análisar e apresentar os resultados.

Como resultados, espera-se obter a prototipagem de um *Smart Mirror* que:

- Seja capaz de reconhecer o usuário posicionado a sua frente.
- Obtenha informações através de APIs e um Banco de Dados local e apresente ao usuário em uma interface no espelho;
- Permita que o usuário personalize informações através de um aplicativo para *smartphone*;
- Permita que o usuário o controle através de gestos de uma das mãos;

Da interface é esperado que:

1. Apresente informações como: data, hora atual e temperatura na cidade definida pelo usuário, notícias, valores atuais de dólar e euro, e itens das listas de tarefas e eventos;
2. Mostre informações personalizadas ao usuário identificado;
3. Permita visualização e exclusão dos itens das listas de eventos e tarefas.

Do aplicativo, espera-se que:

1. Permita ao usuário definir sua cidade atual;
2. Permita mostrar ou omitir uma ou mais informações da interface;
3. Permita adicionar, deletar e editar itens das listas de tarefas e eventos.

Do sistema de controle por gestos, espera-se que:

1. Tenha boa precisão na detecção de mãos e classificação dos gestos;
2. Realize a identificação de gestos com velocidade adequada ao sistema (em poucos segundos), de forma a tornar o controle da interface eficiente.
3. Permita que o usuário navegue pela interface.

Do sistema de controle por fala, espera-se que:

1. Reconheça comandos simples;
2. Tenha velocidade adequada ao sistema: mostre a execução do comando sobre a interface em um espaço de tempo pequeno.;

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as informações teóricas resultantes dos estudos realizados sobre os conceitos nos quais o projeto se baseia.

2.1 INTELIGÊNCIA ARTIFICIAL

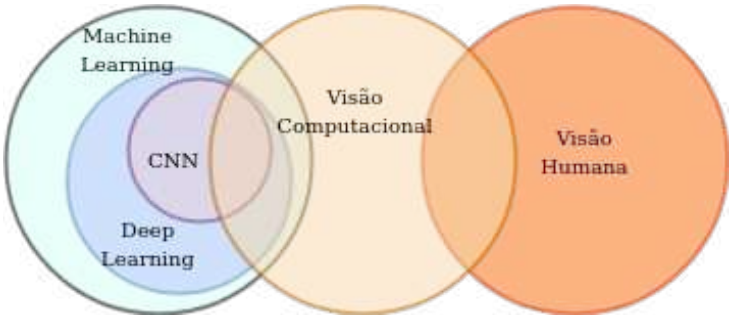
De acordo com BENÍTEZ et al. (2014), Inteligência Artificial é uma disciplina acadêmica criada na década de 1960, que possui relação com a teoria da computação, e que tem como objetivo emular, através de sistemas artificiais, a faculdade intelectual humana; mais especificamente, a capacidade de reconhecimento de padrões. Desde sua criação, suas aplicações mais comuns foram o tratamento de dados e identificação de sistemas, mas ela também conseguiu resolver problemas mais abstratos, como a tradução automática de textos e a demonstração de teoremas matemáticos.

2.1.1 *DEEP LEARNING* E VISÃO COMPUTACIONAL

Segundo BUDUMA e LOCASCIO (2017), pesquisas mais recentes em *Machine Learning* vêm buscando construir modelos inspirados em estruturas utilizadas pelo cérebro humano, para resolver problemas mais complexos. Esse novo campo de pesquisa é comumente chamado *Deep Learning* (DL), Aprendizagem Profunda em português, e tem sido bem sucedido na resolução de problemas envolvendo Visão Computacional e Processamento de Linguagem Natural.

A Figura 1 mostra a relação entre as áreas citadas.

Figura 1 – Diagrama de Venn representando a relação entre *Machine Learning*, *Deep Learning*, CNN, Visão Computacional e Visão Humana



Fonte: Adaptado de KHAN, RAHMANI e ALI (2018)

Machine Learning e, por consequência, *Deep Learning* usam como base e inspiração para criação de seus modelos, o neurônio humano, a célula que constitui uma unidade fundamental do cérebro humano.

BUDUMA e LOCASCIO (2017) explica que cérebro é formado por uma rede de neurônios interconectados entre si. O neurônio é otimizado para receber informações de outros neurônios, através de estruturas chamadas dendritos, processar as informações de um jeito particular e mandar o resultado para outras células. Os dendritos são fortalecidos ou enfraquecidos com base na frequência em que são utilizados. O grau de força do dendrito determina o grau de contribuição da informação de entrada para o resultado do processamento. Após as informações terem sido ponderadas pelo grau de força da conexão, elas são somadas pelo núcleo do neurônio, propagadas por uma estrutura chamada axônio e enviadas para outras células por terminações sinápticas. Em *Deep Learning* o comportamento do neurônio é traduzido para uma estrutura artificial.

Apenas um neurônio não é capaz de resolver problemas complexos, logo, ainda com base na estrutura do cérebro humano, o Aprendizado de Máquina estuda a construção de redes neurais artificiais. Tal como no cérebro, os neurônios são organizados em camadas dentro da rede, e a informação flui de uma camada para a outra. A rede passa a ter uma camada de entrada, uma de saída e camadas intermediárias. Durante o processo de aprendizagem, chamado de treinamento, a informação trafega entre as camadas de neurônios, os pesos são ajustados para que a saída se aproxime do resultado desejado. Esse processo pode ser repetido inúmeras vezes até que se obtenha resultados adequados.

Segundo KHAN, RAHMANI e ALI (2018), com o aumento da capacidade de processamento dos computadores atuais, em termos de memória e velocidade, foi possível criar técnicas de *Machine Learning* que utilizem um conjunto grande de dados de treinamento. Foi possível, também, criar redes neurais com muitas camadas, chamadas redes neurais profundas (*Deep Neural Networks*). *Deep Learning* oferece algumas vantagens sobre técnicas tradicionais de *Machine Learning*: a simplicidade pelo uso de blocos de camadas de redes, que são repetidos diversas vezes para gerar redes grandes; fácil escalabilidade para grandes conjuntos de dados; e transferência de domínio, no sentido de que um modelo treinado para uma tarefa, pode ser aplicado em outra tarefa similar, e até mesmo pode aprender de maneira generalista o suficiente para funcionar em tarefas com dados escassos para treinamento.

Devido ao sucesso alcançado com o uso de redes neurais profundas, técnicas de *Deep Learning* são atualmente consideradas a melhor opção para tarefas de segmentação, detecção, classificação e reconhecimento de objetos em imagens.

2.1.1.1 VISÃO COMPUTACIONAL

Segundo KHAN, RAHMANI e ALI (2018), Visão Computacional é a ciência que busca desenvolver métodos para replicar o sistema visual humano, mais precisamente, sua capacidade de localizar e classificar elementos em imagens, inferindo características do mundo real tridimensional. Ela possui diversas aplicações, tais como interação humano-máquina inteligente, robótica e em multimídia.

Há diversos *frameworks* e plataformas disponíveis voltadas para o uso e desenvolvimento de sistemas de Visão Computacional como por exemplo a *Vision API* do Google Cloud[1] e a biblioteca OpenCV.

2.1.1.2 OPENCV

OpenCV (2015) é biblioteca multiplataforma de software de código aberto voltada para Visão Computacional e *Machine Learning*, construída para fornecer uma infraestrutura comum para aplicações de visão computacional. Ela possui implementações para processamento de imagens e algoritmos de treinamento de classificadores e detectores.

2.1.1.2.1 HAAR-CASCADES

O OpenCV oferece um método de treinamento chamado Haar-Cascades, o qual é muito utilizado em projetos para identificação de faces. Haar-cascade é um método eficiente de detecção de objetos proposto por Paul Viola e Michael Jones, baseado em *Machine Learning*, em que uma função cascata é treinada com diversos exemplos positivos e negativos (no caso de detecção de faces, são respectivamente imagens com rostos e imagens sem rostos).

O Classificador Haar-Cascade é baseado em uma técnica de análise de *pixels* em "quadrados"por função, que se assemelham aos *kernels* utilizados no treinamento de uma CNN, mas com parâmetros definidos manualmente. Haar-Cascade utiliza o algoritmo de aprendizado *Ada-boost* que seleciona um pequeno número de *features* de um grande conjunto para tornar o classificador mais eficiente e então usa técnicas de cascata para detectar a face na imagem.

2.1.1.3 PROCESSAMENTO DE IMAGENS

Para KHAN, RAHMANI e ALI (2018) processamento de imagens é considerado como um passo de pré-processamento para visão computacional, com o objetivo de extrair características importantes da imagem, tais como contornos e cantos, com a manipulação de *pixels* e a aplicação de filtros e morfologia, entre outros. Seu principal objetivo é tratar imagens cruas, de forma a extrair características importantes para um sistema de visão computacional, sem fornecer *feedback* sobre suas características, enquanto que a visão computacional busca produzir descrições semânticas de imagens.

As próximas subseções apresentam técnicas de processamento de imagens utilizadas neste trabalho.

[1] <https://cloud.google.com/vision/docs> (acessado em 30/06/2020)

2.1.1.3.1 SEGMENTAÇÃO BINÁRIA

Uma imagem binária consiste em *pixels* brancos e pretos, apenas. A segmentação binária é o processo de tornar uma imagem binária, que consiste em definir um limiar de segmentação, ou seja, um valor de *pixel* para dividir todos os *pixeis* da imagem entre pretos e brancos. Exemplificando: uma imagem em escala de cinza, de 8 *bits*, possui 256 intensidades de cinza, ou seja, 256 valores possíveis para cada *pixel*. Caso se deseje escolher como limiar o valor de 120, todos os *pixels* com valor acima disso passarão a valer 255 (branco) e todos com valor abaixo passarão a valer 0 (preto). O resultado final será uma imagem com as cores preto e branco somente.

2.1.1.3.2 EQUALIZAÇÃO DE HISTOGRAMA

Histograma é um gráfico de distribuição de frequências, que ajuda a visualizar a distribuição de dados de acordo com a frequência em que eles ocorrem. Para uma imagem em escala de cinza, ele ajuda a visualizar a frequência de *pixels* para cada nível de cinza.

A equalização de histograma é um método que busca redistribuir as frequências de maneira a mudar a forma da distribuição original. O efeito disso em uma imagem em escala de cinza é o ajuste do contraste.

2.1.1.4 REDES NEURAIS CONVOLUCIONAIS

Segundo KHAN, RAHMANI e ALI (2018), Redes Neurais Convolucionais (CNN) são uma categoria de redes neurais profundas que provaram ser muito eficientes em áreas como reconhecimento e classificação de imagens.

O treinamento de CNNs exigiu o desenvolvimento de técnicas para aumentar a eficiência do modelo final. Algumas dessas técnicas são o *Data Augmentation* e o *Transfer Learning*.

2.1.1.5 *DATA AUGMENTATION*

Segundo MIKOŁAJCZYK e GROCHOWSKI (2018), um dos maiores problemas enfrentados durante o treinamento de CNNs é a falta de dados de qualidade, ou desbalanceamento no número de dados entre classes. Uma maneira de lidar com esse problema é utilizando *aumento de dados*, do inglês, *data augmentation*, que consiste na aplicação de técnicas de processamento de imagens para gerar transformações afins e elásticas no conjunto de dados: rotação, reflexão, corte, mudança na paleta de cores, *zoom*, entre outras operações. Na maioria dos casos, essa técnica ajuda a aumentar a performance da rede neural convolucional.

2.1.1.6 *TRANSFER LEARNING*

Uma das técnicas que podem facilitar o treinamento de uma rede neural convolucional é o *Transfer Learning*. Essa técnica consiste em utilizar um modelo pré-treinado, que se sabe ser bom na classificação de um determinado conjunto de dados, e realizar um novo treinamento com os dados desejados, reutilizando e ajustando os parâmetros da rede pré-treinada. Dessa forma, a rede pode reconhecer classes para as quais ela não foi originalmente treinada. Como o treinamento não é realizado a partir do zero, não é necessário utilizar tantos dados e também não é necessário treinar por tanto tempo quanto seria necessário caso contrário.

Segundo PRATT (1993), a ideia de transferência tem origem na Psicologia e é um paradigma padrão em Neurobiologia, em que sinapses muitas vezes vêm pré-conectadas. Dessa forma, o reaproveitamento da rede pré-treinada imita essa ideia.

O *Transfer Learning* é feito retirando-se o conjunto final de camadas (a parte que retorna as inferências) da rede pré-treinada, e colocando no lugar um conjunto de camadas novo, com parâmetros iniciais aleatórios. Todas as camadas abaixo do topo são "congeladas"para que seus parâmetros não sejam alterados, e então a rede é retreinada com uma taxa de aprendizagem bastante baixa. Uma opção também válida é o *Fine-tuning*, que consiste em descongelar o resto da rede para continuar o treinamento, e que pode ajudar a alcançar melhoras significativas na acurácia do modelo. Ao final, obtém-se um modelo generalizado apenas para os últimos dados utilizados.

2.1.1.7 *FRAMEWORKS* PARA *DEEP LEARNING*

Frameworks são abstrações de *software*, ou seja, é *software* criado para oferecer funcionalidade genérica para um determinado propósito, e que pode ser utilizado por desenvolvedores como base para criar aplicações. Eles tornam mais fácil o desenvolvimento de *software* complexo, como, por exemplo, para *Deep Learning*.

Existem diversos *frameworks* voltados para DL, sendo o Tensorflow o mais conhecido e utilizado. O escopo deste projeto envolve a utilização de Tensorflow e Keras.

2.1.1.7.1 TENSORFLOW

Tensorflow (ABADI et al., 2015) um *framework* de código aberto, desenvolvido e mantido pelo Google, com suporte a diversas linguagens de programação, como por exemplo Python, Javascript e C++. Possui documentação ampla e torna a implementação de redes neurais bastante simplificada.

Possui uma versão voltada para implantação de modelos de ML em dispositivos móveis e projetos de IoT, chamada Tensorflow Lite, que funciona com uma grande variedade de dispositivos, desde microcontroladores até *smartphones* potentes. Tensorflow

Lite apresenta também formas de otimizar o modelo para diminuir seu tamanho e sua latência, ou seja, o tempo necessário para se executar uma única inferência com um modelo.

2.1.1.7.2 KERAS

Keras (CHOLLET et al., 2015) é um *framework* construído utilizando como base o Tensorflow, ou seja, executa Tensorflow em seu núcleo, porém oferece mais facilidade para uso, sendo bom para iniciantes na área de DL por precisar de pouco código para implementação e possuir uma curva de aprendizado menor.

2.1.1.8 CONJUNTOS DE DADOS

No contexto de *Deep Learning*, conjuntos de dados precisam ter algumas características para permitirem que o treinamento de um modelo tenha bons resultados. Aqui serão apresentadas algumas dessas características, com o foco em conjuntos de dados de imagens.

O conjunto de dados, mais especificamente de imagens, deve conter exemplares próximos do que se espera que o modelo seja capaz de reconhecer após o treinamento. Para que o modelo consiga obter a melhor generalização possível, ou seja, o melhor aprendizado, é necessário que haja uma quantidade grande de dados e que esses dados apresentem uma grande variação de características entre si. Em geral, quanto maior o número de imagens melhor, no entanto, utilizar mais imagens durante o treinamento implica em um consumo maior de memória RAM do computador.

Fatores como resolução e tipo da imagem (colorida, escala de cinza, preta e branca) também influenciam no resultado do treinamento. Caso as imagens sejam coloridas, o modelo pode aprender que uma cor é uma característica importante para a detecção ou classificação mesmo que ela não seja de fato, como em casos em que apenas o contorno do objeto de interesse é o suficiente.

Outro fator importante a se considerar é o fundo das imagens. Ele pode conter características que serão aprendidas pelo modelo como parte do objeto de interesse da imagem, o que pode causar diminuição da acurácia do modelo. Algumas formas de diminuir a influência do fundo da imagem no treinamento são: utilizar imagens com fundo monocromático e sem detalhes ou fazer remoção do fundo com técnicas de processamento de imagens.

2.2 INTERNET DAS COISAS

Segundo MAGRANI (2018), Internet das Coisas (*Internet of Things*, IoT) se trata da automatização progressiva de setores da economia e da sociedade, baseada na comunicação entre máquinas. Não existe um conceito utilizado unanimemente, mas pode ser entendido como um ambiente em que objetos físicos estão interconectados através da

internet, por meio de sensores, gerando um ecossistema de computação ubíqua. Ou seja, se trata da conexão e interação entre computadores, objetos e sensores, e da forma como eles processam informações, de maneira a facilitar o cotidiano das pessoas. IoT se baseia, portanto em um conjunto de tecnologias e protocolos que permitem essa interação.

2.2.1 DISPOSITIVOS INTELIGENTES

"Um dispositivo inteligente é um dispositivo eletrônico consciente de contexto capaz de desempenhar computação autônoma e se conectar a outros dispositivos com ou sem fio para troca de dados"(SILVERIO-FERNÁNDEZ; RENUKAPPA; SURESH, 2018). Tradução da autora.

Na definição acima se encontram alguns conceitos-chave para se entender dispositivos inteligentes: consciência de contexto, computação autônoma e conectividade. Consciência de contexto pode ser definida como a habilidade de obter informações sobre o ambiente ao redor e ajustar o comportamento de acordo. Computação autônoma é o desempenho de funções de forma autônoma. Conectividade é a habilidade de se conectar a uma rede de dados, sendo ela a característica fundamental para que um dispositivo seja considerado como parte da Internet das Coisas. Portabilidade não é uma qualidade necessária a um dispositivo inteligente.

Dispositivos inteligentes são o que forma a Internet das Coisas e, segundo os conceitos dados, é possível converter os mais diversos objetos em dispositivos *smart* com a adição de sensores, um microcontrolador ou microprocessador, e conectividade.

2.3 BANCO DE DADOS

Segundo DATE (2004), um sistema de banco de dados é um sistema computadorizado cujo objetivo é realizar a manutenção e armazenamento de registros. Tal sistema permite fazer operações sobre os dados nele arquivados, tais como inserção, busca, alteração e remoção de dados. Um banco de dados, por si só, é definido como um repositório para dados computadorizados. Dados podem ser qualquer tipo de informação que seja necessária para auxiliar nas atividades ou processos de um indivíduo ou organização.

ELMASRI e NAVATHE (2005) especifica que os dados são dados com significado implícito, como por exemplo nomes e endereços de pessoas.

Assim, bancos de dados podem conter informações utilizadas por diversos tipos de sistemas que necessitem de dados organizados, classificados e facilmente acessíveis e modificáveis.

2.4 COMUNICAÇÃO *WIRELESS*

TSE e VISWANATH (2005) explica que, ao contrário do que é popularmente dito, a comunicação sem fio é um campo da Telecomunicação existente há mais de 120

anos, tendo sido começado em torno de 1897, com demonstrações de telegrafia sem fio por Marconi. Desde então, surgiram diversos tipos de tecnologia de comunicação sem fio, e muitos deles deixaram de ser utilizados algum tempo depois e foram substituídos por comunicação a cabo, quando essa se mostrava mais eficiente.

O escopo deste trabalho envolve o uso da tecnologia *IEEE 802.11 WLAN*, conhecida popularmente como *Wi-fi*, enquanto tecnologia de comunicação sem fio.

2.4.1 *IEEE 802.11 WLAN*

IEEE 802.11 WLAN usa frequências de rádio para mandar sinais entre dispositivos, utilizando a faixa de frequências entre 2,4GHz e 5GHz. Uma rede local sem fio (WLAN), também conhecida como *Wi-Fi*, é baseada na tecnologia IEEE 802.11 e assim como uma rede local (LAN) segue uma estrutura celular. Cada célula é chamada de "conjunto de serviço básico"(BSS), consiste de nós móveis (MNs) e é controlada por uma estação base, ou ponto de acesso (AP). A maioria das WLANs é formada por diversas células em que os APs são conectados através de um sistema de distribuição (DS), considerado a espinha dorsal, que normalmente utiliza a tecnologia Ethernet, mas também pode ser *wireless*. A WLAN inteiramente conectada, incluindo todas as células, pontos de acesso e sistema de distribuição, também é conhecida como "conjunto de serviço extendido"(ESS). (FITZEK; CHARAF, 2009)

CHHABRA (2013) explica que *Wi-Fi*, *Wireless Fidelity*, utiliza bandas de frequência que podem ser utilizadas por qualquer um em comunicação de rádio. Ele também é a tecnologia com maior taxa de transferência de dados, se comparado a tecnologias como Bluetooth e ZigBee, e possui diversos métodos de segurança disponíveis. ELKHODR, SHAHRESTANI e CHEUNG (2016) explica que essa é uma tecnologia que tem um consumo de energia mais alto e que por isso não é ideal para aplicações de IoT que dependem do uso de baterias.

2.5 *SINGLE-BOARD COMPUTERS*

Um *single-board computer* (SBC), computador de placa única, é um computador completo construído em uma única placa de circuito, que compreende microprocessador, memória, interfaces de entrada e saída assim como um computador pessoal comum. Porém, diferentemente de um computador comum, um *single-board computer* não utiliza expansões para outras funções; isso reduz o custo do sistema.

Há diversos tipos de computadores de placa única disponíveis no mercado, utilizando uma grande variedade de microprocessadores, bem como com uma grande variedade de capacidade de processamento. Eles também apresentam muitas vantagens em sua utilização: são mais compactos, mais leves e mais eficientes no uso de energia que computadores com diversas placas. Apesar de todas essas vantagens, eles também possuem diversas limi-

tações, como por exemplo o número pequeno de interfaces de entrada e saída. Além disso, a maioria dos SBCs possuem pouca quantidade de memória e utilizam microprocessadores menos potentes.

SBCs são muito utilizados em aplicações embarcadas, bem como para robótica, projetos de Internet das Coisas e automação domiciliar.

Três exemplos de SBCs são brevemente apresentados abaixo, em ordem crescente de poder de processamento, com seus respectivos valores em dólares americanos. É possível através deles, demonstrar a grande variedade existente entre SBCs.

- **Raspberry Pi 4 Model B - 4GB**: é o SBC mais popular e muito utilizado para ensino de conceitos de computação básica em escolas, e também em projetos de Internet das Coisas domésticos e até mesmo industriais. É vendido por cerca de $55,00. A Figura 2 mostra como ele é.
  O modelo *4 Model B* é o mais recente e poderoso existente durante a realização deste trabalho.
  Abaixo estão algumas de suas especificações técnicas:
  - Memória RAM LPDDR4 de 4 GB.
  - Processador BCM2711B0 Broadcom, quad-core Cortex-A72 com clock de 1.5 GHz.
  - Duas portas USB 2.0 e duas USB 3.0
  - Porta Gigabit Ethernet

Figura 2 – Raspberry Pi 4 Model B

Fonte: Página do Raspberry Pi[2]

- **NVIDIA Jetson Nano**: é o modelo mais básico de sua linha, porém mais poderoso que o Raspberry Pi apresentado anteriormente, por possuir uma Unidade de Processamento Gráfico (GPU). Foi desenvolvido para permitir a execução de redes neurais com bom desempenho. Tem suporte da NVIDIA JetPack, que inclui um pacote de suporte para placa (BSP), Sistema Operacional Linux, e bibliotecas de software NVIDIA CUDA, cuDNN, e TensorRT^{TM} para *deep learning* e visão computacional, entre outros. É vendido por algo em torno de $99.00. A Figura 3 mostra o SBC.
  Algumas de suas especificações técnicas são:

- GPU Maxwell, 128-core
- Memória de 4 GB, 64-bit LPDDR4 25.6 GB/s
- Processador Quad-core ARM A57 @ 1.43 GHz
- Quatro portas USB 3.0, USB 2.0 Micro-B
- Porta Gigabit Ethernet

Figura 3 – NVIDIA Jetson Nano

Fonte: Página da NVIDIA Jetson Nano[3]

- **Udoo Bolt V8**: é o modelo mais poderoso entre os apresentados. Permite a execução de jogos, mineração de criptomoedas, execução de modelos de Visão Computacional e Inteligência Artificial. É vendido por cerca de $420.00. A Figura 4 apresenta a placa.
  Especificações técnicas:
  - GPU Amd Radeon™ Vega 8 Graphics
  - Dois *sockets* de memória DDR4 Dual-channel 64-bit.
  - Processador AMD Ryzen™ Embedded V1605B Quad Core/oito threads, 2.0GHz.
  - Duas portas USB 3.0 Type-A e duas USB Type-C
  - Porta Gigabit Ethernet

2.6 *APPLICATION PROGRAMMING INTERFACE*

Segundo SOUZA et al. (2004) *Application Programming Interface* (API), interface de programação de aplicação, é uma interface bem definida que determina o serviço que um componente de *software* provê a outros componentes, e que permite que um componente de *software* acesse outro componente programaticamente.

SOUZA et al. (2004) também explica que a palavra interface é usada para indicar explicitamente que APIs são construídas nos limites de pelo menos duas aplicações de *software* diferentes. APIs oferecem flexibilidade e proteção contra mudanças, ajudando a impedir que mudanças em um componente de *software* afetem outro que depende dele. Elas são utilizadas na indústria para ocultar detalhes de implementação para desenvolvedores

Figura 4 – Udoo Bolt V8

Fonte: Página do Udoo Bolt[4]

e dividir o desenvolvimento de *software* distribuído, e são consideradas o único jeito de construir sistemas de componentes semi-dependentes, de maneira escalável.

Existe uma grande diversidade de APIs disponíveis para as mais diferentes necessidades e aplicações, devido à comodidade que seu uso proporciona.

3 METODOLOGIA

Este capítulo apresenta o método proposto para atingir os objetivos definidos anteriormente na Seção 1.1.

3.1 ANÁLISE E MODELAGEM DO PROJETO

A análise ocorreu com a definição exata das partes do projeto, através dos seguintes passos:

1. Análise das partes do projeto;
2. Análise dos melhores opções de hardware e software para implementação para cada parte do projeto;
3. Análise das funções desejadas para cada parte do projeto.

A modelagem foi feita a partir dos resultados obtidos na análise:

1. Levantamento dos requisitos do sistema;
2. Levantamento dos recursos necessários;
3. Modelagem da base de dados;
4. Modelagem do aplicativo Android;
5. Modelagem da interface do espelho;
6. Modelagem do sistema de controle;
7. Protótipo das telas do aplicativo;
8. Protótipo da interface;
9. Protótipo da estrutura física.

3.2 DESENVOLVIMENTO E TESTES

Após a etapa de modelagem, iniciou-se a etapa de desenvolvimento do sistema, composta dos seguintes passos:

1. Aquisição dos recursos necessários;
2. Criação da base de dados e servidor;
3. Desenvolvimento dos *scripts* da interface do servidor para consulta ao banco SQL;
4. Desenvolvimento da interface do espelho;
5. Desenvolvimento do aplicativo Android;
6. Desenvolvimento do sistema de controle por gestos;
7. Desenvolvimento do sistema de controle por voz;
8. Integração do sistema de gestos com a interface;

9. Otimização do sistema de gestos.

3.3 CONCLUSÃO

A etapa de conclusão ocorreu com a análise dos resultados e testes finais do sistema. Houve também a conclusão dos pontos positivos e negativos do sistema.

4 ANÁLISE E MODELAGEM DO PROJETO

Este capítulo explica como se deu a análise e como as decisões foram tomadas para a estrutura de cada parte do projeto.

4.1 LEVANTAMENTO DOS RECURSOS NECESSÁRIOS

Todas as partes do sistema, com exceção do aplicativo, devem funcionar em um *single-board computer*, que possa ser colocado dentro da estrutura do espelho. O *single-board computer* deve ser utilizado por ser compacto e por ter um custo baixo. Através de pesquisas sobre outros projetos de *Smart Mirror*, foi possível perceber que a maioria deles utiliza um Raspberry Pi para implementar o espelho.

O projeto, porém, propõe-se a utilizar *Deep Learning* para implementar o controle por gestos, que apesar de ser uma boa opção pela performance, consome uma quantidade alta de recursos, principalmente memória e processamento. O ideal, portanto, seria utilizar um computador com bom poder de processamento. Porém, mais alguns fatores foram levados em consideração para a escolha do computador central do projeto, além do poder de processamento: preço e facilidade de aquisição. Na comparação entre alguns modelos, foi possível perceber que o principal fator para escolha seria a facilidade de aquisição, pois a maioria deles não são encontrados com facilidade em lojas brasileiras e precisariam ser importados, o que é pouco conveniente devido ao tempo de importação. Considerando os fatores citados, o modelo escolhido foi o Raspberry Pi 4 Model B.

Apesar de não ser um computador *single-board* construído para o uso de Inteligência Artificial, como é por exemplo o Jetson Nano da NVIDIA, o Raspberry Pi 4 tem poder de processamento para executar um modelo de *Deep Learning*, bem como o suporte do *framework* Tensorflow para ajudar na otimização da execução, o que o torna uma escolha aceitável. Além disso, é de fácil aquisição e de valor mais baixo.

Além do computador, o projeto exige para a sua execução, a aquisição de um monitor, uma *webcam*, um microfone, além de cabos e adaptadores.

4.2 MODELAGEM E REQUISITOS

Nesta Seção são apresentados os resultados da modelagem e os requisitos funcionais e não funcionais para as partes do sistema com as quais o usuário deverá interagir mais diretamente.

4.3 BANCO DE DADOS

Nesta Seção é apresentada a lógica de modelagem do banco de dados. Na Seção 4.8, encontra-se o Diagrama Entidade Relacionamento produzido a partir da modelagem.

4.3.1 MODELAGEM

Para o projeto, é necessário armazenar dados de usuários, bem como informações referentes às listas de tarefa e eventos para cada usuário. É necessário haver uma tabela para dados de usuários: id, nome, cidade, e informações que deseja exibir na interface. Também é necessário haver uma tabela para tarefas e outra para eventos, ambas precisam armazenar: id, id do usuário e descrição.

4.4 APLICATIVO ANDROID

Nesta Seção são apresentados a modelagem e os requisitos funcionais e não funcionais para o aplicativo.

4.4.1 MODELAGEM

O aplicativo deve apresentar uma interface amigável para que o usuário possa ver e editar suas preferências na interface, bem como adicionar, excluir ou editar itens nas listas de tarefas e eventos. Decidiu-se criar três telas: uma para configuração da cidade do usuário e das informações exibidas na interface, uma para edição de uma lista de tarefas e outra para edição de uma lista de eventos.

4.4.2 REQUISITOS FUNCIONAIS PARA O APLICATIVO

RF01 O aplicativo deve permitir exibir ou ocultar informações da interface;
RF02 O aplicativo deve permitir a edição da cidade do usuário;
RF03 O aplicativo deve permitir a adição, edição e exclusão de itens das listas de tarefas e eventos.

4.4.3 REQUISITOS NÃO FUNCIONAIS PARA O APLICATIVO ANDROID

RNF01 O aplicativo deve ser de uso simples;
RNF02 O aplicativo deve ser desenvolvido para o sistema operacional Android.

4.5 SISTEMAS DE CONTROLE POR GESTOS E FALA

Nesta Seção é descrita a lógica de modelagem para os dois sistemas de controle, por gestos e fala, e apresentados os requisitos funcionais e não funcionais para ambos.

4.5.1 MODELAGEM

O sistema de controle por gestos deve ter boa precisão na detecção de gestos, além disso não pode ser muito lento pois isso tornaria o uso do espelho pouco eficiente. Não é possível evitar certo grau de latência no seu funcionamento, mas devem ser utilizados meios de otimizá-lo.

O reconhecimento de gestos deve ser feito em duas partes: detecção de mãos e classificação da mão detectada. As duas etapas visam permitir que o usuário tenha mais liberdade para posicionar a mão em frente ao espelho para realizar os gestos. O classificador deve ser capaz de reconhecer pelo menos seis gestos para permitir uma interação adequada com a interface.

O sistema de controle por voz deve ser preciso e simples, com o uso de poucas palavras para comandos. Assim como no controle por gestos, haverá certo grau de latência, mas deve ser otimizado.

4.5.2 REQUISITOS FUNCIONAIS PARA O SISTEMA DE CONTROLE POR FALA

RF01 O sistema de controle por voz deve reconhecer um número pequeno de comandos simples;

RF02 O sistema de controle por voz deve permitir navegar pela interface e excluir itens das listas de tarefas e eventos;

4.5.3 REQUISITOS NÃO FUNCIONAIS PARA O SISTEMA DE CONTROLE POR FALA

RNF01 O sistema de controle por fala ter boa precisão;

RNF02 O sistema de controle por fala não deve ter muita latência no funcionamento;

RNF03 O sistema de controle por fala deve ser composto de comandos simples.

4.5.4 REQUISITOS FUNCIONAIS PARA O SISTEMA DE CONTROLE POR GESTOS

RF01 O sistema de controle por gestos deve realizar detecção da mão em frente ao espelho e classificação do gesto;

RF02 O sistema de controle por gestos deve ser capaz de classificar seis gestos diferentes;

RF02 O sistema de controle por gestos deve ser capaz de identificar a movimentação da mão para cima, para baixo, para a esquerda e para a direita;

4.5.5 REQUISITOS NÃO FUNCIONAIS PARA O SISTEMA DE CONTROLE POR GESTOS

RNF01 O sistema de controle por gestos deve ter boa precisão;

RNF02 O sistema de controle por gestos não deve ter latência muito grande no reconhecimento dos gestos.

4.6 INTERFACE

Nesta Seção são apresentados os requisitos funcionais e não funcionais para a interface do espelho, bem como a lógica de modelagem.

4.6.1 MODELAGEM

A interface do espelho deve ter aspecto simples e agradável, disponibilizando as principais informações e um menu de acesso às informações mais detalhadas. Ela também deve apresentar uma mensagem de boas-vindas ao usuário, quando o mesmo for identificado, e após isso apresentar a tela principal com as informações personalizadas; e deve reagir aos comandos de voz e fala do usuário. Se o usuário usar o comando de saída, deve apresentar uma mensagem de despedida.

4.6.2 REQUISITOS FUNCIONAIS PARA A INTERFACE

RF01 A interface deve exibir informações de maneira simples;
RF02 A interface deve exibir informações: hora e data, tempo, notícias, valor do euro e dólar, menu, ícones e um item da lista de tarefas e de eventos;
RF03 A interface deve ser passível de interação através de gestos de fala do usuário.

4.6.3 REQUISITOS NÃO FUNCIONAIS PARA A INTERFACE

RNF01 A interface deve ter aparência agradável.

4.7 DESIGN DA ESTRUTURA FÍSICA

A estrutura do espelho foi pensada para ser composta por madeira e possuir a frente em vidro, com insulfilm, que imita um espelho, mas permite e a visualização da interface. O espaço interno deve comportar um monitor, bem como todos os cabos, computador e periféricos. A *webcam* e o microfone devem ficar escondidos dentro da estrutura, de forma a tornar sua aparência estéticamente mais agradável.

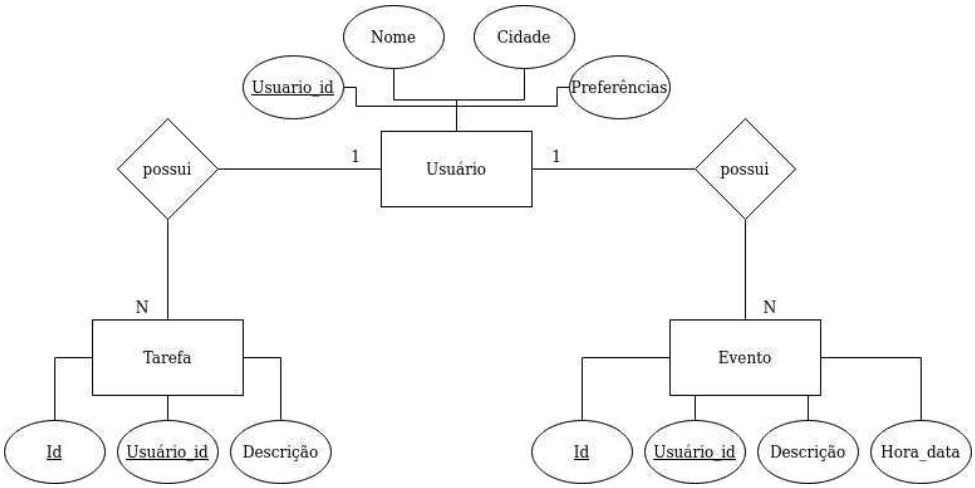
4.8 DIAGRAMAÇÃO

Nesta Seção são apresentados o Diagrama Entidade Relacionamento do banco de dados criado e o Diagrama de Caso de Uso do sistema proposto.

4.8.1 DIAGRAMA ENTIDADE-RELACIONAMENTO DO BANCO DE DADOS

Foi criado um Diagrama Entidade Relacionamento (DER), apresentado na Figura 5, para demonstrar a estrutura do Banco de Dados, as tabelas e inter-relacionamentos necessários para o projeto.

Figura 5 – Diagrama entidade-relação da estrutura do banco de dados.



Fonte: Autoria própria

4.8.2 DIAGRAMA DE CASOS DE USO

A Figura 6 apresenta os casos de uso do aplicativo móvel e da interface do espelho. Os casos são descritos a seguir.

4.8.2.1 APLICATIVO

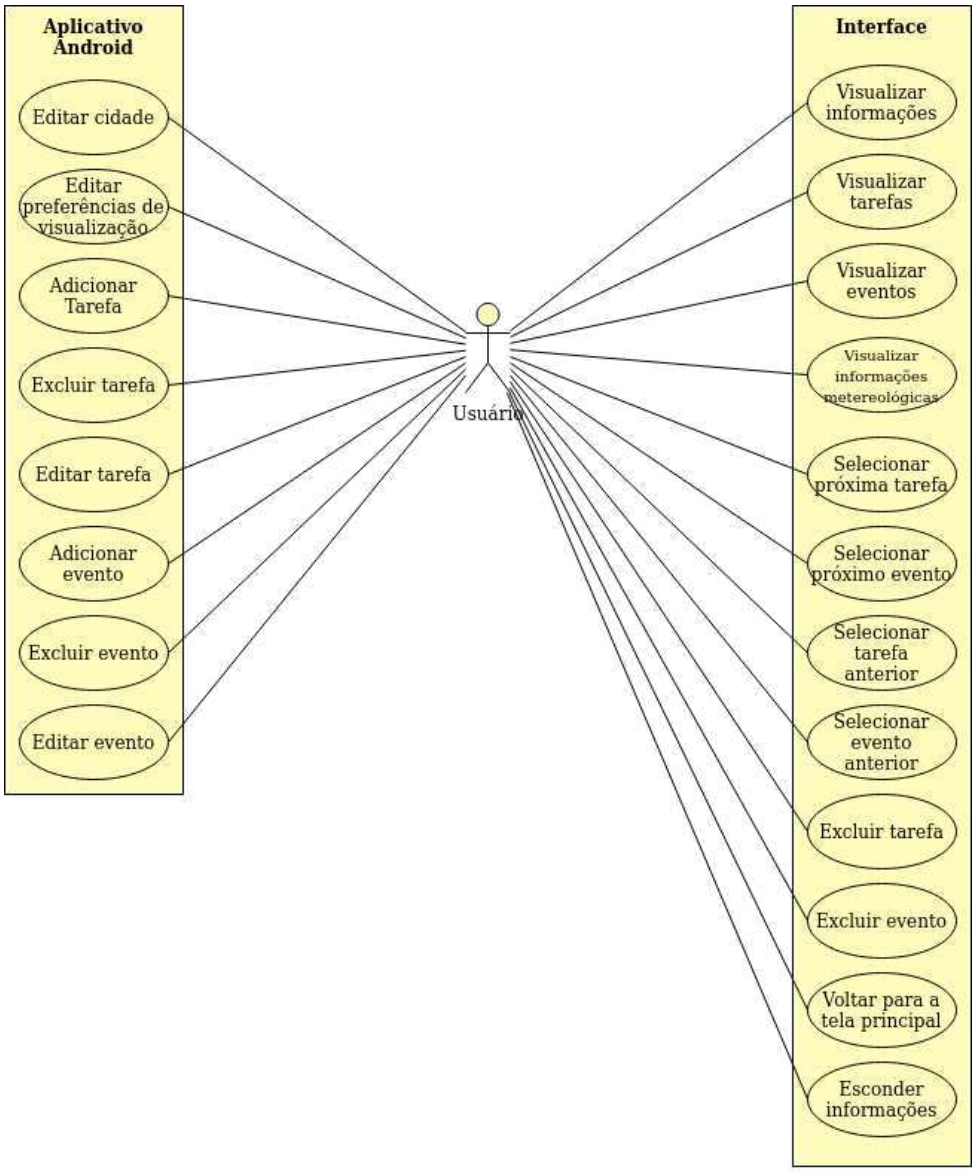
Casos de uso para o aplicativo:

Caso: **Editar cidade**
Descrição: o usuário deseja alterar a cidade configurada
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal: na tela inicial, o usuário edita o campo com o nome da cidade e em seguida aperta o botão "Salvar configuração"para salvar a alteração no banco de dados.
Fluxo alternativo: se o campo de nome da cidade estiver vazio, aparecerá um alerta solicitando o preenchimento.

Caso: **Editar preferências de visualização**
Descrição: o usuário deseja alterar os tipos de informação que são exibidos na interface
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal: na tela inicial, o usuário pode selecionar ou desselecionar as caixa de opções. Em seguida aperta o botão "Salvar configuração".

Caso: **Adicionar tarefa**
Descrição: o usuário deseja adicionar uma tarefa à lista de tarefas
Pré-condições: o usuário está conectado à internet e autenticado

Figura 6 – Diagrama de casos de uso.



Fonte: Autoria própria

Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de tarefas". O aplicativo muda para a tela de tarefas;
2. O usuário então aperta o botão de sinal de "mais"no canto inferior direito da tela. Um campo aparece na tela para o preenchimento da descrição da tarefa;
3. O usuário então aperta o botão "ok"debaixo do campo para salvar a tarefa.

Fluxos alternativos:

3. Se o usuário quiser cancelar a adição, basta apertar o botão "cancelar"e o campo de preenchimento será ocultado;
4. Se o usuário não preencher o campo e apertar em "ok", um alerta surgirá

requisitando o preenchimento.

Pós-condições: o campo de preenchimento da tarefa é ocultado e então é exibida a tela de tarefas com a nova tarefa adicionada.

Caso: **Editar tarefa**
Descrição: o usuário deseja editar uma tarefa da lista de tarefas
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de tarefas". O aplicativo muda para a tela de tarefas;
2. O usuário então aperta o botão ao lado da tarefa desejada e escolhe a opção editar. Um campo aparece na tela para a alteração da descrição da tarefa;
3. O usuário faz a alteração e então aperta o botão "ok"debaixo do campo para salvar a tarefa.

Fluxos alternativos:

3. Se o usuário quiser cancelar a edição, basta apertar o botão "cancelar"e o campo de preenchimento será ocultado;

Pós-condições: o campo de preenchimento da tarefa é ocultado e então é exibida a tela de tarefas com a tarefa atualizada.

Caso: **Excluir tarefa**
Descrição: o usuário deseja editar uma tarefa da lista de tarefas
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de tarefas". O aplicativo muda para a tela de tarefas;
2. O usuário então aperta o botão ao lado da tarefa desejada e escolhe a opção excluir;
3. A tarefa é excluída.

Pós-condições: é exibida a tela de tarefas atualizada.

Caso: **Adicionar evento**
Descrição: o usuário deseja adicionar um evento à lista de eventos.
Pré-condições: o usuário está conectado à internet e autenticado.
Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de eventos". O aplicativo muda para a tela de eventos;
2. O usuário então aperta o botão de sinal de "mais"no canto inferior direito da tela. Aparecem na tela os campos para preenchimento da descrição do evento, data e hora;
3. O usuário então aperta o botão "ok"debaixo do campo para salvar o evento.

Fluxos alternativos:

3. Se o usuário quiser cancelar a adição, basta apertar o botão "cancelar"e o campo de preenchimento será ocultado;
4. Se o usuário não preencher algum dos campos e apertar em "ok", um alerta surgirá requisitando o preenchimento.

Pós-condições: os campos de preenchimento do evento são ocultados e então é exibida a tela de tarefas com o novo evento adicionado.

Caso: **Editar evento**
Descrição: o usuário deseja editar um evento da lista de eventos
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de eventos". O aplicativo muda para a tela de eventos;
2. O usuário então aperta o botão ao lado do evento desejado e escolhe a opção editar. Os campos aparecem na tela para a alteração dos dados do evento (hora, data e descrição);
3. O usuário faz a alteração e então aperta o botão "ok"debaixo do campo para salvar o evento.

Fluxos alternativos:

3. Se o usuário quiser cancelar a edição, basta apertar o botão "cancelar"e o campo de preenchimento será ocultado;

Pós-condições: os campos de edição do evento são ocultados e então é exibida a tela de eventos com o evento atualizado.

Caso: **Excluir evento**
Descrição: o usuário deseja editar um evento da lista de eventos.
Pré-condições: o usuário está conectado à internet e autenticado
Fluxo principal:

1. Na tela inicial, o usuário aperta o botão "Lista de eventos". O aplicativo muda para a tela de eventos;
2. O usuário então aperta o botão ao lado do evento desejado e escolhe a opção excluir;
3. O evento é excluído.

Pós-condições: é exibida a tela de eventos atualizada.

4.8.2.2 INTERFACE

Casos de uso para a interface do espelho:

Caso: **Visualizar informações**

Descrição: o usuário deseja visualizar as informações do espelho.
Fluxo principal:

1. O usuário se posiciona em frente ao espelho e espera ser reconhecido.
2. Após o reconhecimento, aparecerá a frase de boas vindas e então a tela inicial
3. Na tela inicial, o usuário poderá visualizar todas as principais informações

Caso: **Visualizar tarefas**
Descrição: o usuário deseja visualizar as tarefas na interface
Pré-condições: o usuário foi reconhecido pelo espelho
Fluxo principal:

1. Na tela inicial, o usuário realiza o movimento para a esquerda ou para a direita com a mão, até selecionar o ícone de tarefa no menu. Então realiza o gesto de "mão aberta"para exibir a tela de tarefas. Alternativamente, o usuário pode usar o comando de fala "tarefas".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: a lista de tarefas é exibida

Caso: **Visualizar eventos**
Descrição: o usuário deseja visualizar os eventos na interface
Pré-condições: o usuário foi reconhecido pelo espelho
Fluxo principal:

1. Na tela inicial, o usuário realiza o movimento para a esquerda ou para a direita com a mão, até selecionar o ícone de evento no menu. Então realiza o gesto de "mão aberta"para exibir a tela de eventos. Como alternativa, pode ser utilizado o comando de fala: "eventos".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: a lista de eventos é exibida

Caso: **Visualizar informações meteorológicas**
Descrição: o usuário deseja visualizar informações meteorológicas na interface.
Pré-condições: o usuário foi reconhecido pelo espelho.
Fluxo principal:

1. Na tela inicial, o usuário realiza o movimento para a esquerda ou para a direita com a mão, até selecionar o ícone de meteorologia no menu. Então realiza o gesto de "mão aberta"para exibir a tela com as informações meteorológicas. O usuário também pode utilizar o comando de fala: "tempo".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: a lista de informações meteorológicas é exibida.

Caso: **Selecionar próxima tarefa**
Descrição: o usuário deseja selecionar a tarefa de baixo da lista.
Pré-condições: o usuário está visualizando a tela de tarefas
Fluxo principal:

1. O usuário realiza o movimento para baixo com a mão ou utiliza o comando de fala "baixo".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: o elemento de baixo é selecionado

Caso: **Selecionar o próximo evento**
Descrição: o usuário deseja selecionar o evento de baixo da lista.
Pré-condições: o usuário está visualizando a tela de eventos.
Fluxo principal:

1. O usuário realiza o movimento para baixo com a mão. Também é possível utilizar o comando de fala: "baixo".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: o elemento de baixo é selecionado.

Caso: **Selecionar o evento anterior**.
Descrição: o usuário deseja selecionar a tarefa de cima da lista.
Pré-condições: o usuário está visualizando a tela de tarefas
Fluxo principal:

1. O usuário realiza o movimento para cima com a mão. O comando de fala equivalente é "cima"e também pode ser utilizado.

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento ou falar o comando novamente.

Pós-condições: o elemento de cima é selecionado

Caso: **Selecionar o evento anterior**.
Descrição: o usuário deseja selecionar o evento de cima da lista.
Pré-condições: o usuário está visualizando a tela de eventos.
Fluxo principal:

1. O usuário realiza o movimento para cima com a mão. Alternativamente, o comando "cima"pode ser pronunciado pelo usuário

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento ou comando de fala.

Pós-condições: o elemento de cima é selecionado.

Caso: **Excluir tarefa**.
Descrição: o usuário deseja excluir a tarefa selecionada.
Pré-condições: o usuário está visualizando a tela de tarefas
Fluxo principal:

1. O usuário realiza o gesto de "mão fechada"ou pronuncia "apagar".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento.

Pós-condições: o elemento é excluído e a lista aparece atualizada

Caso: **Excluir evento**.
Descrição: o usuário deseja excluir o evento selecionado.
Pré-condições: o usuário está visualizando a tela de eventos
Fluxo principal:

1. O usuário realiza o gesto de "mão fechada"ou pronuncia "apagar".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento ou comando.

Pós-condições: o elemento é excluído e a lista aparece atualizada

Caso: **Voltar para a tela principal**.
Descrição: o usuário deseja voltar para a tela de informações principais.
Pré-condições: o usuário está visualizando a tela de tarefas ou de eventos.
Fluxo principal:

1. O usuário realiza o gesto de "mão aberta". Como alternativa, o usuário pode pronunciar a palavra "voltar".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento ou comando.

Pós-condições: a interface mostra a tela principal.

Caso: **Esconder informações**.
Descrição: o usuário deseja ocultar as informações do espelho.
Pré-condições: o usuário foi reconhecido pelo espelho.

Fluxo principal:

1. O usuário realiza o gesto de "saudação vulcana"ou pronuncia "sair".

Fluxos alternativos:

1. O sistema de reconhecimento de gestos não reconheceu o movimento. O usuário deve tentar repetir o movimento ou comando.

Pós-condições: a interface mostra a mensagem de despedida e oculta todas as informações.

4.9 RESULTADO DA ANÁLISE

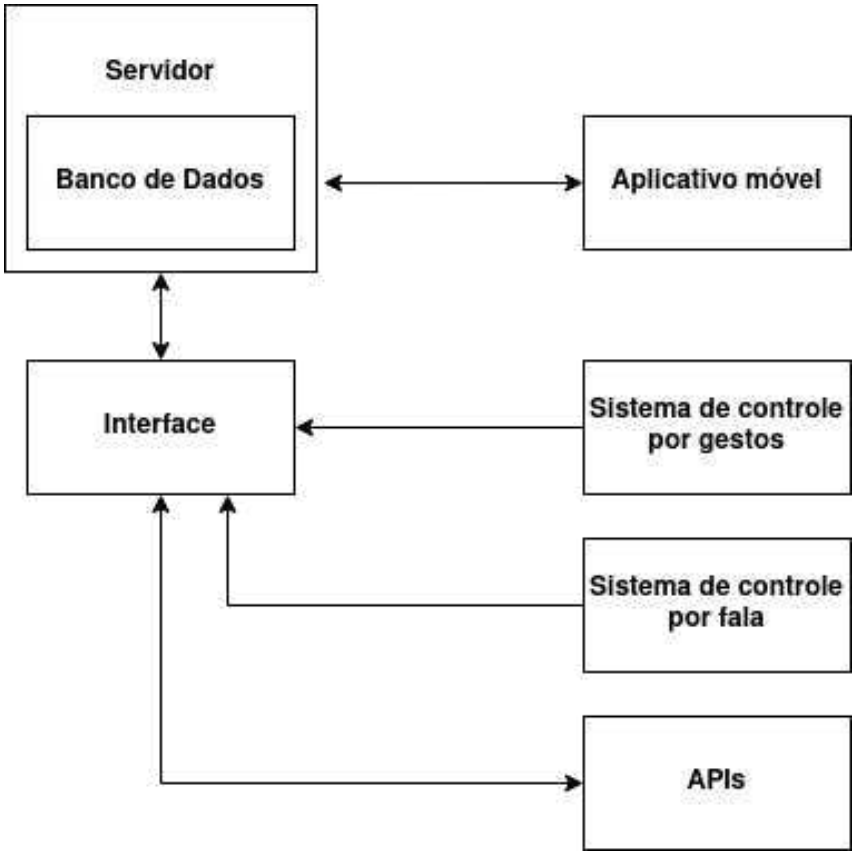
Com a análise, foi possível detalhar quais são as principais partes do sistema e quais as suas funções, além de algumas decisões de implementação:

- **Banco de Dados**: necessário para guardar as informações dos usuários do espelho, além de permitir facilidade na manipulação e organização das informações;
- **Servidor para hospedagem do banco de dados**: necessário para facilitar o uso do banco de dados através de uma interface;
- **Interface do espelho**: parte responsável por exibir as informações ao usuário;
- **APIs**: parte das informações (metereologia, cotações de moedas, noticias, etc) a serem exibidas no espelho devem ser obtidas através de APIs disponíveis na Internet;
- **Sistema de controle por gestos**: permite interagir com o espelho sem toques ou som, portanto pode ser considerada uma solução interessante. Poderia ser implementado por sensores ou uso de inteligência artificial, porém a segunda opção se mostrou melhor por dar mais liberdade ao usuário. Escolheu-se realizar a implementação com Tensorflow e Keras, pela quantidade de documentação e exemplos existentes *online*;
- **Sistema de controle por fala**: é uma alternativa ao controle por gestos e um meio de tornar o projeto mais inclusivo;
- **Aplicativo móvel**: permite ao usuário personalizar as informações exibidas na interface. Escolheu-se implementar o aplicativo para Android, por ser o sistema operacional móvel mais utilizado no mundo[1];
- **Comunicação *wireless***: necessária para permitir o uso do aplicativo com o resto do sistema. Escolheu-se utilizar *Wi-fi*.

A Figura 7 apresenta um diagrama demonstrando as partes do projeto e suas conexões.

[1] <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> (acessado em 30/06/2020)

Figura 7 – Diagrama das partes do projeto.



Fonte: Autoria própria

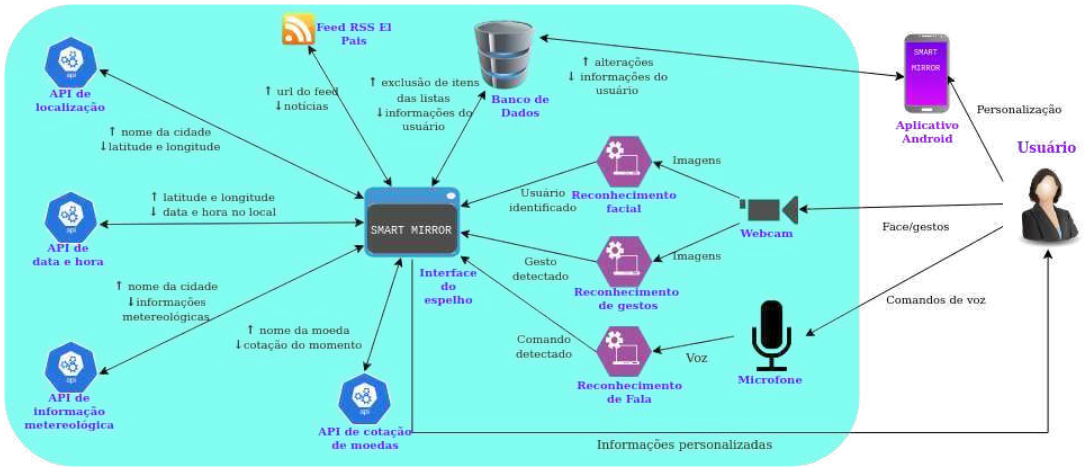
5 DESENVOLVIMENTO E TESTES

Este capítulo apresenta a arquitetura do projeto e as tecnologias utilizadas no desenvolvimento. Apresenta também a descrição das etapas de desenvolvimento de cada parte do projeto e da integração entre elas.

5.1 ARQUITETURA DO PROJETO

A Figura 8 apresenta a arquitetura do projeto. Nela, um usuário é reconhecido pelo sistema através de sua face. A imagem do rosto do usuário passa por um processo que o identifica e manda um comando para a interface do espelho, que por sua vez acessa APIs, um *feed RSS* e um banco de dados local para obter e exibir as informações. O usuário pode, então, interagir com o espelho através de gestos e fala, os quais passam por processos que os identificam e mandam comandos específicos à interface para manipular a visualização das telas. Além disso, o usuário dispõe de um aplicativo Android que permite personalizar as informações exibidas através de consultas ao banco de dados.

Figura 8 – Diagrama da arquitetura do projeto.



Fonte: Autoria própria

5.2 TECNOLOGIAS UTILIZADAS

As tecnologias utilizadas no desenvolvimento são brevemente descritas e listadas nesta Seção.

Para o desenvolvimento da interface foram utilizados:

- Express (versão 4.17.1)[1]: é um *framework* para desenvolvimento de aplicações *web* em Node.js;
- Electron (versão 7.1.2)[2]: é um *framework* para criação de aplicações *desktop* utilizando tecnologias *web* como JavaScript, HTML e CSS.

Para o desenvolvimento do sistema de reconhecimento de gestos, foram utilizados:

- Tensorflow V2 (classificador);
- API de detecção de objetos do Tensorflow; V1[3]
- Keras.

Foram utilizados para o sistema de reconhecimento facial:

- Python (versão 3.7);
- OpenCV (versão 3.4.6).

Para o banco de dados, o aplicativo Android e o sistema de reconhecimento de fala, foram utilizados:

- MySQL;
- Kotlin[4]: é uma linguagem de programação multiplataforma, orientada a objetos e funcional, utilizada para o desenvolvimento de aplicativos para Android;
- Python (versão 3.7).

Foram utilizadas algumas APIs para obter informações para o espelho. Elas são listadas abaixo:

- Nominatim[5]: utilizada para obter a latitude e longitude da cidade configurada pelo usuário;
- TimeZoneDB[6]: utilizada para obter a hora e data da cidade;
- OpenWeather[7]: utilizada para obter dados metereológicos da cidade configurada;
- Exchange rates[8]: utilizada para obter as cotações das moedas dólar e euro, em reais.

5.3 RECURSOS DE *HARDWARE* UTILIZADOS

Foram utilizados os recursos listados abaixo:

- Raspberry Pi 4 com 4 GB de memória RAM;

[1] <https://expressjs.com/> (acessado em 30/06/2020)
[2] <https://www.electronjs.org/> (acessado em 30/06/2020)
[3] <https://github.com/tensorflow/models/tree/master/research/object_detection> (acessado em 30/06/2020)
[4] <https://kotlinlang.org/> (acessado em 30/06/2020)
[5] <https://nominatim.org> (acessado em 30/06/2020)
[6] <https://timezonedb.com/> (acessado em 30/06/2020)
[7] <https://openweathermap.org/> (acessado em 30/06/2020)
[8] <https://exchangeratesapi.io/> (acessado em 30/06/2020)

- Monitor LED 19,5";
- *Webcam*;
- Modulo de Câmera para Raspberry Pi V2;
- Microfone *usb*.

5.4 DESENVOLVIMENTO DA INTERFACE GRÁFICA

A interface gráfica foi inspirada na plataforma MagicMirror, porém bastante simplificada em comparação. Assim como a plataforma citada, o desenvolvimento da interface foi realizado em Javascript, HTML e CSS, utilizando o *framework* Express para aplicativos *web* em Node JS, além do *framework* Electron, que permite desenvolver aplicações para *desktop* com componentes de *front-end* e *back-end*, como aplicações *web*. Ela foi construída como uma *Single Page Application*, ou seja, uma única página *web* que pode ter seu conteúdo modificado conforme as ações do usuário, porém dentro de uma aplicação para *desktop*. A comunicação com os algoritmos de reconhecimento facial e reconhecimento de gestos é feita através de requisições HTTP para a API definida na aplicação. Foram definidos *endpoints* (pontos de acesso definidos na API) específicos para o recebimento de gestos, comandos de fala, chegada e saída do usuário.

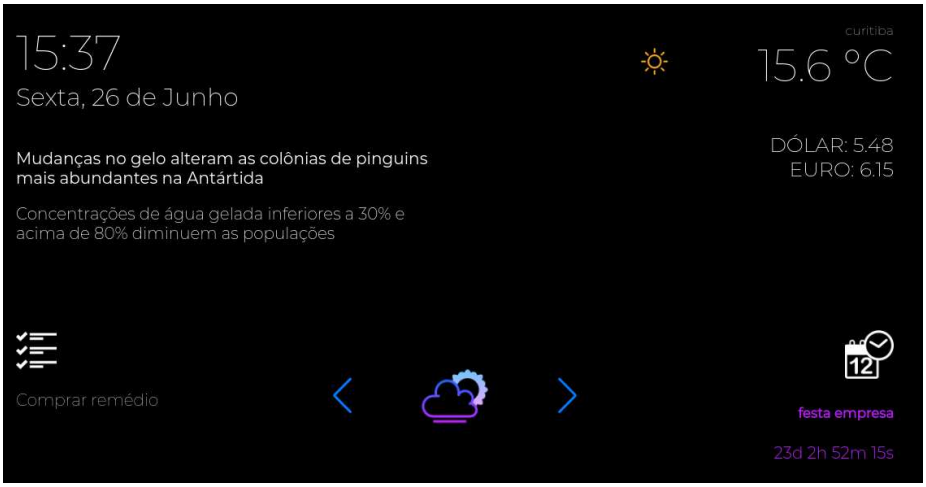
Ela possui uma tela principal com um total de seis informações principais: data e hora, temperatura na cidade especificada, notícias (retiradas do feed RSS do jornal El País Brasil), valores de cotação para euro e dólar, o ícone e um item da lista de tarefas trocado a cada 5 segundos, e um ícone e um item da lista de eventos que também é trocado a cada 5 segundos. Além disso possui o menu de seleção. As informações de data, hora e temperatura utilizam a cidade definida pelo usuário e são disponibilizadas mesmo para cidades fora do Brasil.

A interface é iniciada em uma tela sem informações. Quando um usuário se posiciona em frente ao espelho e é identificado, a aplicação busca todos os seus dados no banco de dados local, realiza as chamadas de APIs, e mostra a mensagem de boas vindas, para então mostrar a tela de informações. A partir daí, ela passa a aceitar os comandos de fala e gestos. A Figura 9 apresenta as telas da interface. Existe um total de cinco telas: a tela de boas vindas, a tela de informações principais, as telas de lista de tarefas e lista de eventos, e a tela de informações meteorológicas que disponibiliza dados atuais e de previsão do tempo para os próximos cinco dias. A tela de eventos mostra o tempo restante para cada evento, atualizado a cada segundo. A Figura 10 identifica os elementos da tela de informações principais.

A interface é feita para reagir a comandos específicos de fala e gestos do usuário, ou seja, existe uma reação para cada comando possível, de acordo com a tela atual que o usuário está visualizando no momento do comando. É possível expandir a lista de tarefas e de eventos ou as informações meteorológicas através de gestos feitos com as mãos (movimentos para a esquerda ou direita para selecionar a opção no menu da tela

Figura 9 – Telas da interface: (a) Tela de boas vindas, (b) Tela principal, (c) Tela da lista de tarefas, (d) Tela da lista de eventos e (e) Tela com informações meteorológicas.

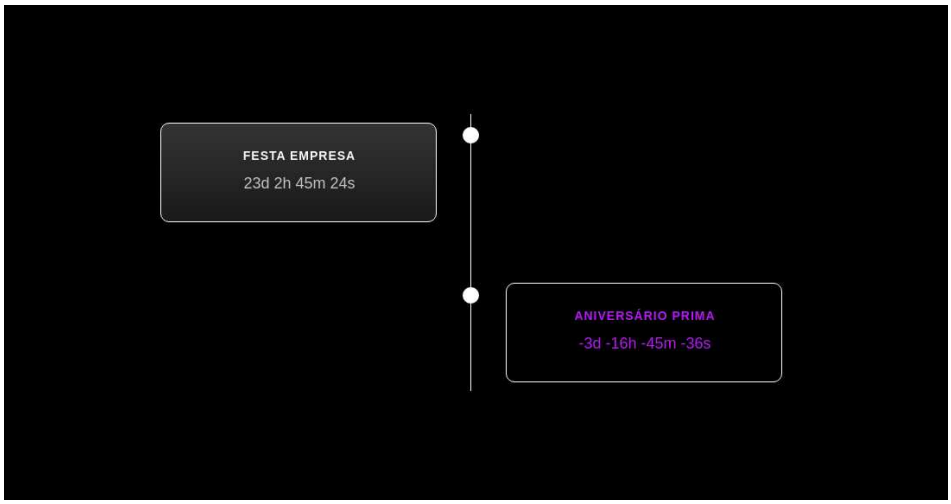
(a)



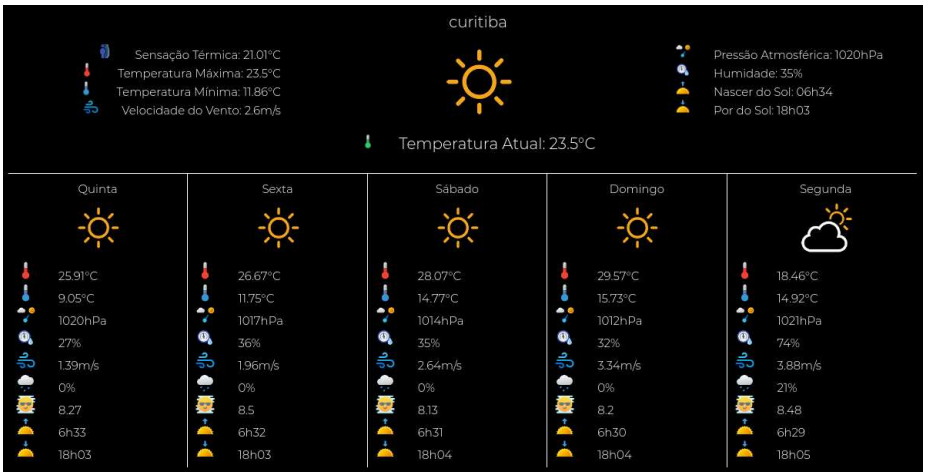
(b)

(c)

Figura 9 – Telas da interface: (a) Tela de boas vindas, (b) Tela principal, (c) Tela da lista de tarefas, (d) Tela da lista de eventos e (e) Tela com informações meteorológicas.



(d)



(e)

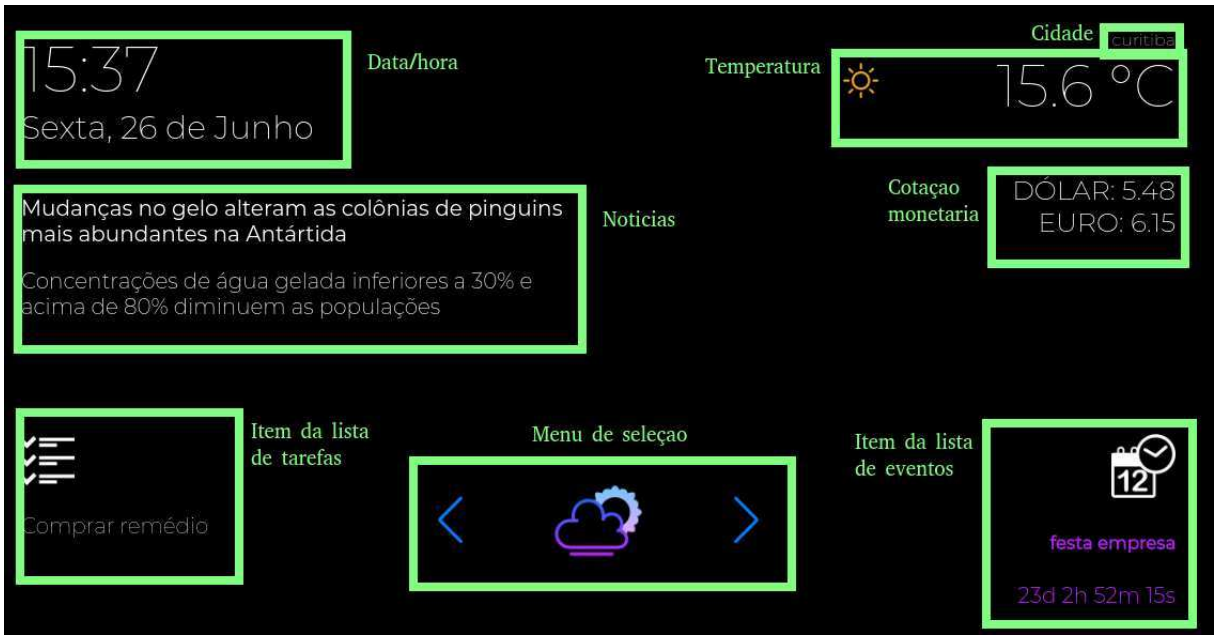
Fonte: Autoria própria

principal, gesto de "mão aberta"para expandir e gesto de "mão aberta"para voltar para a tela principal). Além disso, é possível excluir itens das listas com o gesto de "mão fechada". Caso deseje, o usuário pode também encerrar o uso do espelho, escondendo todas as informações através do gesto de "saudação vulcana". Após isso, o sistema ficará em modo ocioso durante três minutos, para então voltar a reconhecer usuários.

5.5 DESENVOLVIMENTO DO APLICATIVO ANDROID

O usuário pode comunicar e gerenciar informações relevantes ao espelho utilizando um aplicativo Android instalado em seu *smartphone*. O aplicativo desenvolvido na linguagem Kotlin conecta-se ao servidor hospedado no Raspberry Pi através da rede *Wi-fi* local. Para isso, ele procura pelo Raspberry Pi como um *Service* na rede local, através

Figura 10 – Tela principal da interface explicada.



Fonte: Autoria própria

de um nome configurado no próprio Raspberry Pi. Para usar o aplicativo, o usuário deve informar seu nome e senha numérica que são armazenados no banco de dados. Quando o usuário tenta fazer *login*, o aplicativo busca as informações do usuário, através do nome inserido no aplicativo, e compara a senha fornecida com a senha obtida do banco; se elas forem iguais, o usuário tem acesso às outras telas e caso contrário, recebe uma mensagem de falha.

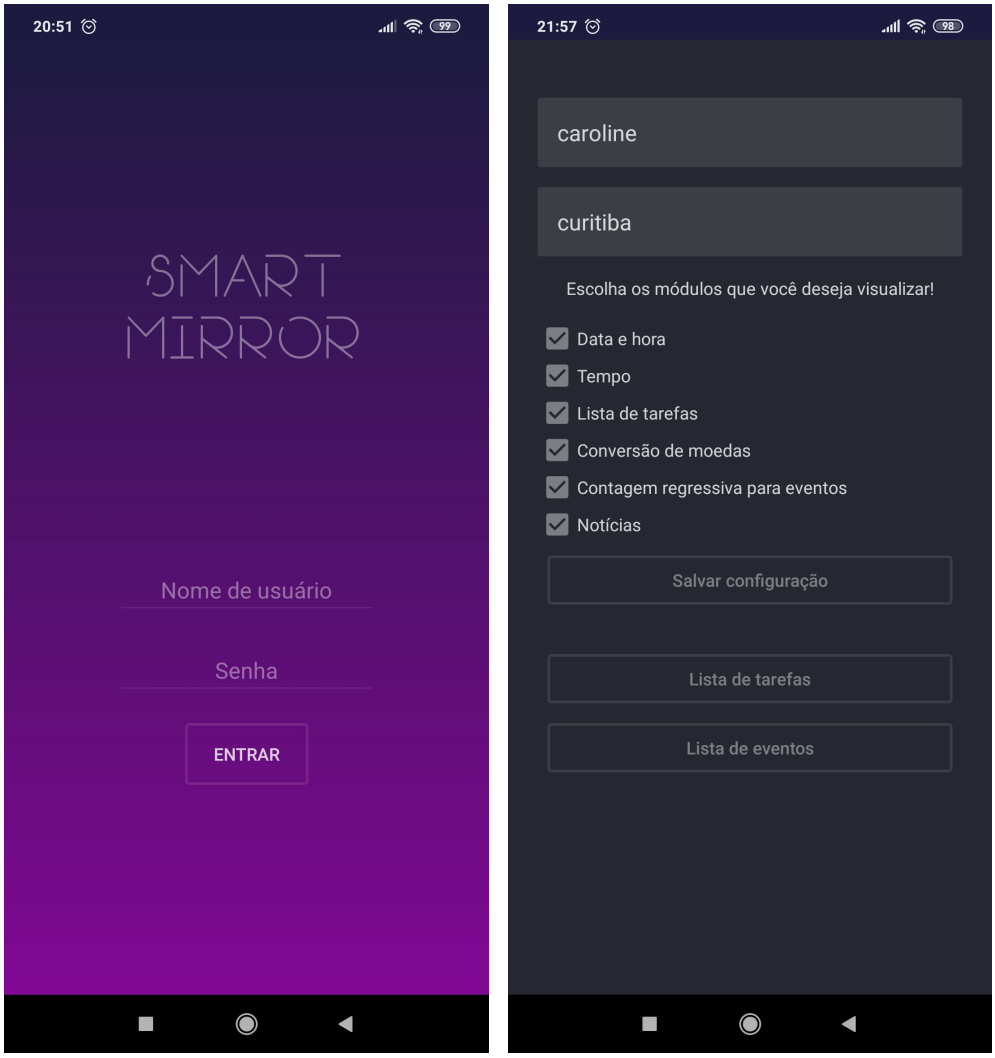
O desenvolvimento do aplicativo Android foi realizado utilizando-se a plataforma Android Studio do Google, um ambiente voltado para a criação de aplicativos. A linguagem escolhida foi o Kotlin, que apesar de recente enquanto linguagem, ganhou bastante espaço entre desenvolvedores de aplicativos Android nos últimos anos por permitir o desenvolvimento mais rápido e com menos linhas de código. O Kotlin pode também ser utilizado para desenvolvimento *web* e multiplataforma móvel.

O aplicativo permite que o usuário digite o nome da cidade, defina a lista de tarefas e a lista de eventos, com inserção, edição e remoção de itens, e também escolha as informações que gostaria de ver no espelho, podendo omitir as que desejar através de "caixas de checagem". Quaisquer alterações nas configurações ou listas são salvas automaticamente no banco de dados local. Cada item da lista de eventos ou da lista de tarefas permite edição e exclusão. Quando há alguma falha de comunicação com o servidor ou algum erro no uso do aplicativo, uma mensagem de erro é apresentada ao usuário.

A Figura 11 apresenta as telas do aplicativo desenvolvido. Ao todo são quatro telas: a tela de *login*, a tela de personalização da cidade e das preferências de visualização

de itens do espelho, e as telas de edição da lista de tarefas e da lista de eventos.

Figura 11 – Telas do aplicativo: (a) Tela de Login, (b) Tela de configurações, (c) Tela da lista de tarefas, (d) Tela da lista de eventos e (e) Tela da lista de tarefas mostrando o menu para editar e deletar.



(a) (b)

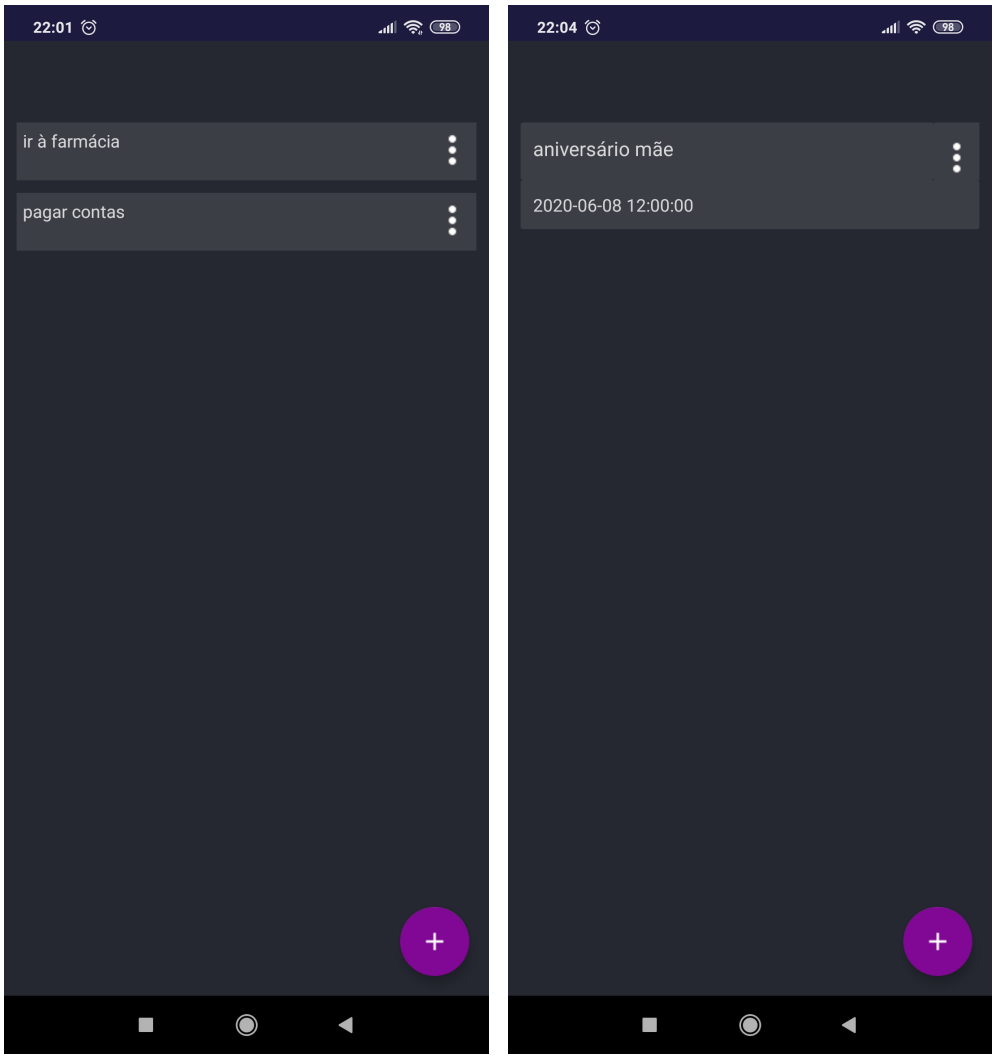
Fonte: Autoria própria

A Figura 12 explica melhor as ações possíveis no aplicativo através dos botões e telas existentes.

5.6 DESENVOLVIMENTO DO SERVIDOR E DO BANCO DE DADOS

Foi desenvolvido um servidor local Apache e banco MySQL no Raspberry Pi para permitir o armazenamento dos dados dos usuários e integrar as diferentes partes do sistema que compõem o espelho. Nele são armazenados os dados do usuário e os dados fornecidos por ele ao aplicativo: sua cidade, lista de eventos e lista de tarefas.

Figura 11 – Telas do aplicativo: (a) Tela de Login, (b) Tela de configurações, (c) Tela da lista de tarefas, (d) Tela da lista de eventos e (e) Tela da lista de tarefas mostrando o menu para editar e deletar.



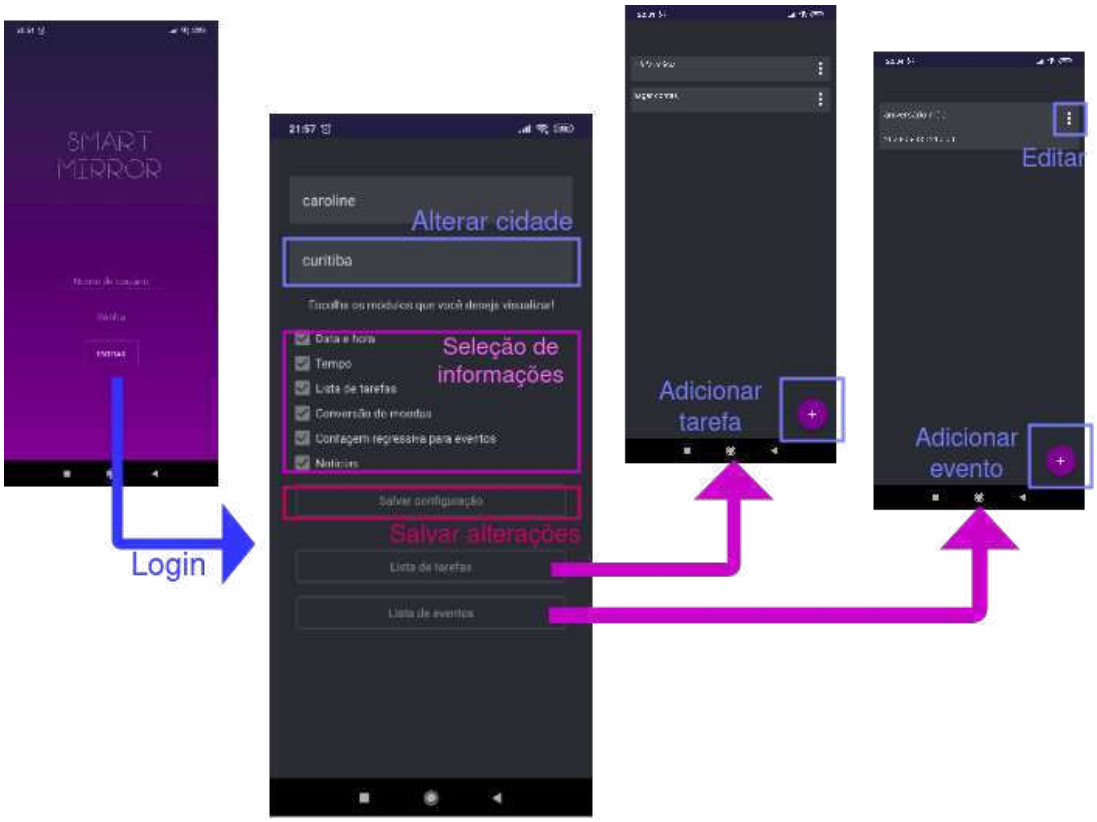
(c) (d)

Fonte: Autoria própria

5.7 DESENVOLVIMENTO DO RECONHECIMENTO FACIAL

O reconhecimento facial foi implementado utilizando o algoritmo Haar Cascades fornecido pelo *framework* OpenCV, que é um modelo de detecção de faces frontais. Para desenvolver o reconhecimento facial, foi necessário fornecer ao Haar Cascades os *encodings* obtidos através das imagens das faces, nos quais determina-se as *features* do rosto do usuário. O treinamento exige que se utilize um conjunto de imagens do rosto de cada um dos usuários do espelho. Para que se obtenha um treinamento robusto, é necessário ter um bom número de imagens, com a maior variação de poses e iluminações possível, para cada usuário. Quanto maior o número de imagens, maior também é o tempo necessário para treinamento e melhores são os resultados. Foram obtidas 50 imagens da autora para

Figura 12 – Telas do aplicativo: (a) Tela de Login, (b) Tela de configurações, (c) Tela da lista de tarefas, (d) Tela da lista de eventos e (e) Tela da lista de tarefas mostrando o menu para editar e deletar.



Fonte: Autoria própria

treinar e testar o algoritmo, com resultados satisfatórios. Porém, caso houvesse mais de uma face a ser reconhecida, é possível que fossem necessárias mais imagens para obter uma boa precisão de reconhecimento.

Durante o funcionamento do espelho, o algoritmo obtém as imagens da câmera, realiza a detecção de face, de forma igual ao processo realizado no treinamento, e então compara com a base de usuários criada previamente durante o treinamento. Dessa comparação, o algoritmo retorna o nome do usuário que mais se assemelha ao detectado, bem como um valor de confiança da classificação, entre 0 e 1. Através de testes, determinou-se um valor mínimo de confiança para estabelecer se o usuário foi de fato identificado.

Para que o sistema de reconhecimento facial ficasse mais robusto, o algoritmo desenvolvido utiliza 5 imagens do usuário, obtém a classificação para cada uma delas e retorna o nome do usuário que apareceu mais vezes entre as classificações.

Como opção alternativa ao uso do algoritmo Haar Cascades, seria possível treinar um modelo em *Deep Learning* para realizar o reconhecimento facial, porém esse modelo consumiria mais recursos computacionais. Como o Haar Cascades apresentou bons resultados e consumo de poucos recursos, ele foi escolhido como a melhor opção para o

projeto.

5.8 DESENVOLVIMENTO DO RECONHECIMENTO DE GESTOS

Há diversas técnicas para reconhecimento de gestos que utilizam Processamento Digital de Imagens (PDI), *Deep Learning* ou uma combinação de ambos. Utilizando apenas PDI, o reconhecimento é feito utilizando técnicas como segmentação binária da imagem e remoção de fundo, entre outras, e após isso obtenção dos contornos na imagem. Esse método foi testado, porém demonstrou muita sensibilidade a variações na iluminação do ambiente, e portanto pouca robustez.

Devido a isso, decidiu-se fazer o reconhecimento dos gestos associando o Processamento Digital de Imagens ao *Deep Learning*. Primeiramente, tentou-se utilizar o treinamento de um modelo de classificação com imagens binárias, as quais eram obtidas com as técnicas de PDI mencionadas e apresentadas ao classificador treinado com imagens semelhantes. Esse processo se mostrou mais robusto que o primeiro, porém ainda bastante sensível à luz, pois para obter a imagem a ser enviada para o classificador treinado, esse deveria passar pela binarização que depende bastante da iluminação do ambiente.

As duas formas de reconhecimento citadas acima poderiam funcionar bastante bem para casos em que a mão fica parada em uma área específica da imagem, em que o fundo da imagem é imutável e com poucos detalhes, já que o conhecimento dessa área possibilita que apenas ela seja analisada e diminui os casos de falsos positivos, porém não poderia ser considerado o ideal para o desenvolvimento do projeto, já que isso traria bastante limitação ao usuário e ao posicionamento do espelho.

Considerando os aspectos de que o usuário deveria poder movimentar a mão livremente e de que modelos utilizando imagens binárias são bastantes sensíveis à luminosidade, decidiu-se desenvolver uma forma de reconhecimento de gestos que utilizasse a detecção de mãos como primeira etapa, seguida da classificação do gesto. Para o treinamento do classificador, escolheu-se utilizar imagens em escala de cinza dos gestos desejados, por ter como vantagens a "ausência"de cores e não necessitar da técnica de segmentação binária. Imagens coloridas também poderiam ter sido utilizadas, porém seria necessário que elas representassem os mais variados tons de pele para que o modelo pudesse ser robusto e inclusivo.

O desenvolvimento do algoritmo final de reconhecimento de gestos foi feito em duas partes: a detecção de mãos e, após o essa etapa, a classificação dos gestos. O modelo de detecção de mãos permitiu que além de gestos estáticos, fosse possível utilizar também a movimentação da mão para controlar o espelho.

5.8.1 TREINAMENTO MODELO DE DETECÇÃO DE MÃOS

Para o modelo de detecção de mãos em uma imagem, foi utilizada a API de reconhecimento de objetos do Tensorflow, juntamente com a técnica de *transfer learning*. Foi utilizado para o treinamento da detecção de mãos o *dataset* Egohands [9], que disponibiliza 4800 imagens de mãos, e o modelo pré-treinado (disponibilizado pelo Tensorflow [10]) SSD MobileNet V2 pré-treinado com o *dataset* Coco 2017. O treinamento foi feito em 200.000 épocas com taxa de aprendizagem de 0.004, em cerca de 25 horas no computador pessoal da desenvolvedora do projeto, com as seguintes especificações:

- 32GB de memória RAM
- Processador AMD Ryzen 7 3800X 3.9GHz
- GPU NVIDIA GeForce RTX 2060 Super 8GB

A GPU utilizada possui *Tensor Cores* [11], que tornam o treinamento de modelos do Tensorflow mais rápido.

O modelo de detecção de mãos terminou o treinamento com o valor de perda de validação (*validation loss*: função de perda para os casos de classificação errada durante a validação do modelo) em torno de 3.2, mas apresentou bons resultados com testes práticos. A Figura 13 apresenta algumas curvas de treinamento obtidas através da ferramenta Tensorboard, que mostram os valores de *loss* durante o treinamento.

A detecção de mãos permitiu que fosse feita a detecção da movimentação da mão em quatro sentidos: para cima, para baixo, para a esquerda e para a direita. Para isso, obtém-se a posição da mão em um *frame* e compara-se com a posição dela no próximo *frame*.

A Figura 14 mostra dois casos de detecção de mão. Através das imagens, é possível perceber que o fundo da imagem não atrapalha na detecção.

5.8.2 TREINAMENTO DO MODELO DE CLASSIFICAÇÃO DE GESTOS

Decidiu-se treinar um classificador que pudesse identificar os seguintes gestos de mão: "*hang loose*", "positivo", "palma da mão", "mão fechada", "paz e amor"e "saudação vulcana". Além dessas classes, foi necessário ter a classe "desconhecido", para evitar falsas classificações.

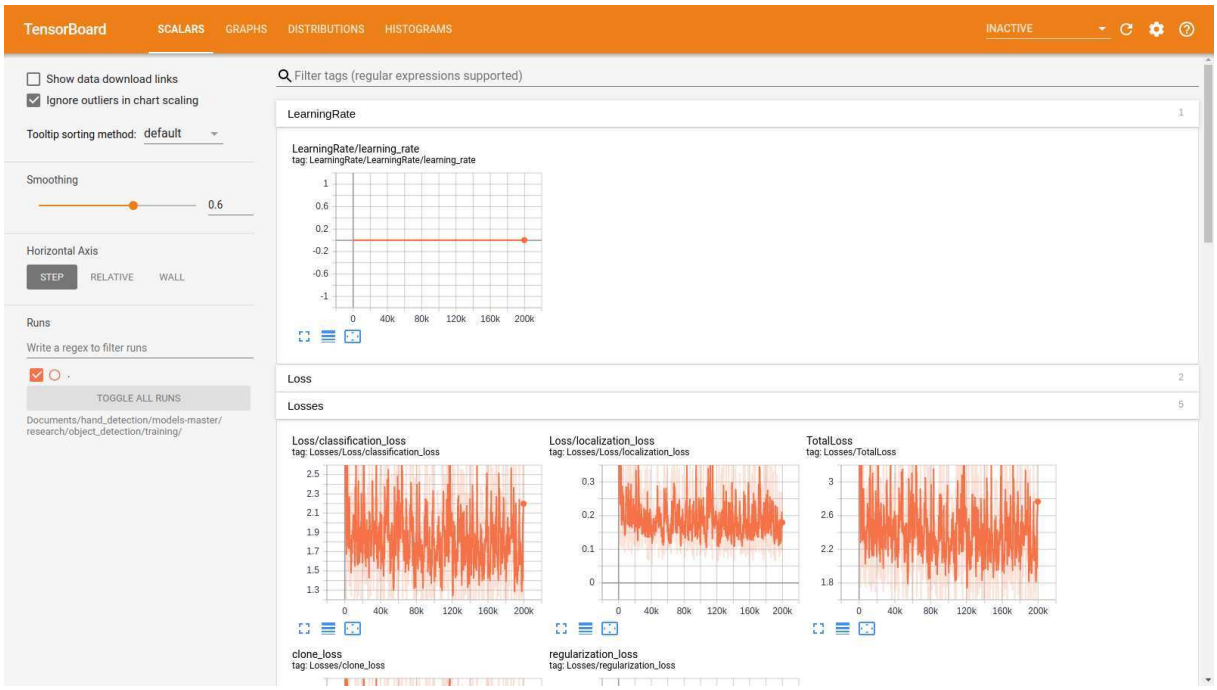
Para o treinamento do classificador, era necessário um conjunto de dados bastante completo que possibilitasse à rede conseguir classificar o gesto para imagens obtidas em ambientes com diferentes tipos de iluminação e com fundos diversos. Para atender a essa necessidade, foi criado pela autora um conjunto de dados com 2000 imagens para cada um

[9] <http://vision.soic.indiana.edu/projects/egohands/> (Acessado em 30/06/2020)

[10] <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md> (Acessado em 30/06/2020)

[11] <https://developer.nvidia.com/tensor-cores> (Acessado em 30/06/2020)

Figura 13 – Métricas de treinamento.



Fonte: Autoria própria

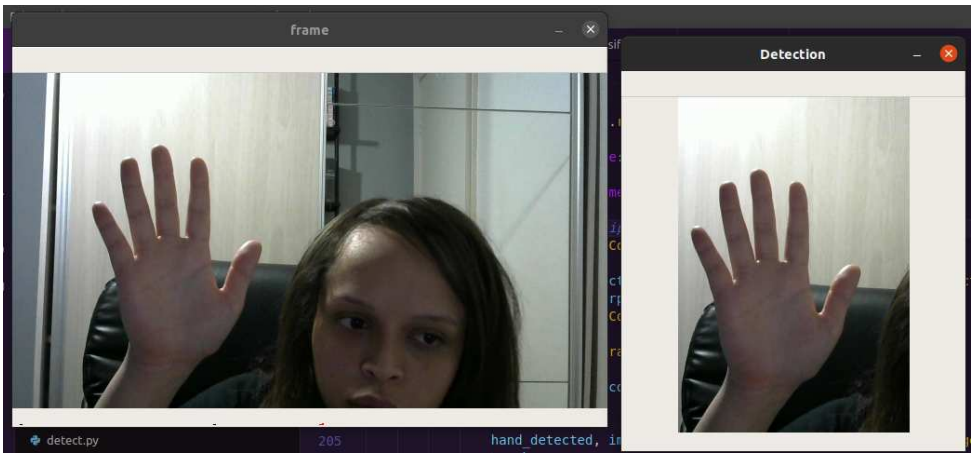
dos gestos possíveis. Todas as imagens foram produzidas utilizando como modelo a mão da autora, em escala de cinza, sempre com o tamanho 200x200 e em diferentes ambientes, com diferentes luminosidades, diferentes posições da mão e com fundos monocromáticos. Sobre cada uma das imagens foi aplicada a equalização de histograma com a função implementada no OpenCV, para aumentar o contraste das imagens e evidenciar a área da mão em cada uma delas. A Figura 15 ilustra alguns exemplos de imagens do conjunto de dados criado.

O efeito esperado com o uso de imagens produzidas em escala de cinza e em diferentes luminosidades é que gestos em diferentes cores de pele possam ser igualmente classificados. Como pode ser visto na Figura 15, a variação de luminosidade faz a mão da autora (de cor branca), aparecer com diversos tons diferentes, o que poderia ajudar o modelo a classificar corretamente outras cores de pele futuramente. Não foi possível fazer testes com pessoas de outras cores de pele para saber se os resultados são de acordo com os esperados.

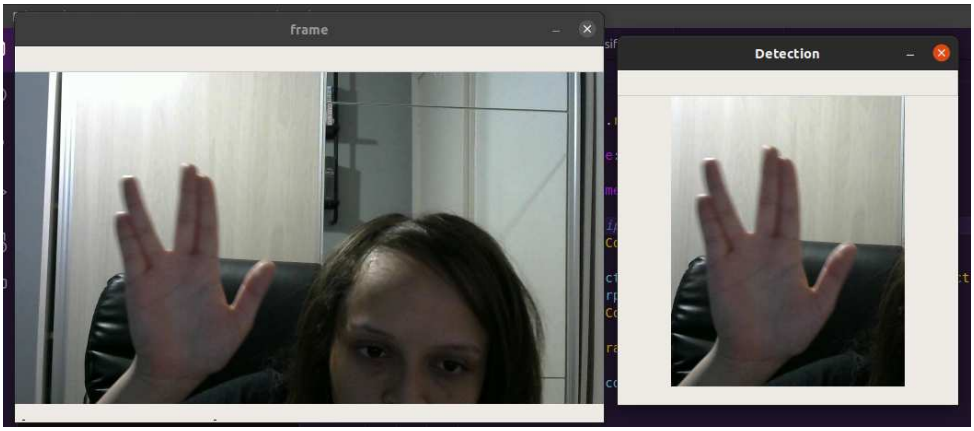
O treinamento do classificador foi feito com o auxílio das técnicas de *data augmentation* e *transfer learning*, utilizando a arquitetura *ResNet*. Também foi realizado o *fine tuning* da rede. A escolha da arquitetura foi feita entre os modelos fornecidos pela API Keras [12], com base no tamanho do modelo e na acurácia (quanto menor o tamanho do modelo, mais rápido ele é carregado na memória, e quanto maior a acurácia, melhor é

[12] <https://keras.io/api/applications/> (Acessado em 30/06/2020)

Figura 14 – Imagens mostrando dois casos de detecção de mão com gestos diferentes: (a) Detecção de mão com gesto de "palma"e (b) Detecção de mão com gesto de "spock".



(a)



(b)

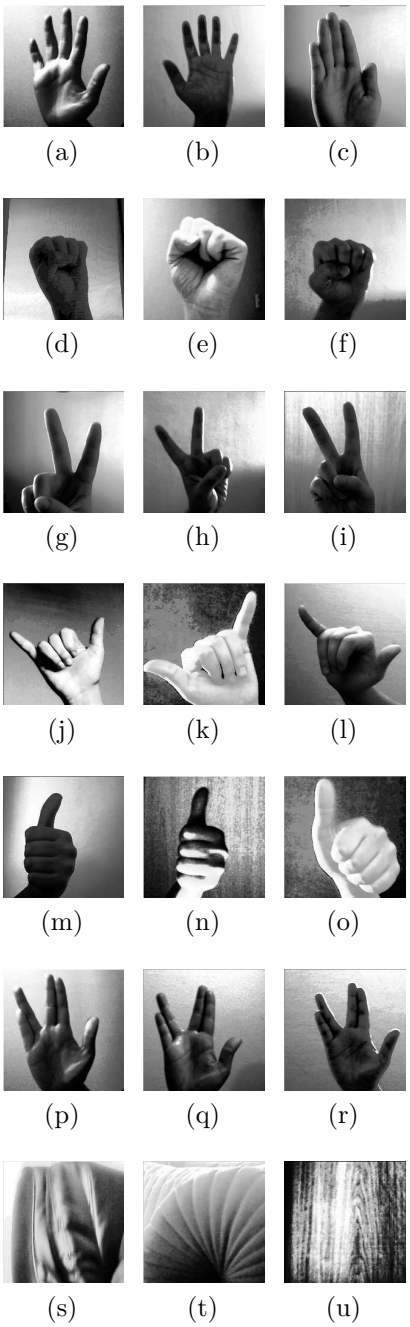
Fonte: Autoria própria

seu desempenho). Considerando tais requisitos, o modelo ResNet era uma das melhores opções na época em que foi escolhido o modelo de treinamento.

A técnica de *transfer learning* foi utilizada para auxiliar no treinamento, pois o número de imagens produzidas (2000 para cada classe) é teoricamente o mínimo necessário para se obter um modelo com boa generalização; então o *transfer learning*, como técnica que ajuda a treinar modelos quando não se tem um conjunto grande de imagens, foi utilizado como forma de tentar obter um melhor resultado.

O classificador de gestos foi treinado em cerca de 6 horas, no mesmo computador descrito na Seção 5.8.1, com taxa de aprendizagem de 0.0004, lote de 200 imagens e em 90 épocas. Na Figura 16 estão presentes uma imagem com os valores de métricas de treinamento do classificador para as três últimas épocas, e também os gráficos com as curvas de treinamento. O primeiro gráfico mostra a etapa de *transfer learning* em que ocorre a maior parte do aprendizado, como pode ser observado. O segundo gráfico mostra

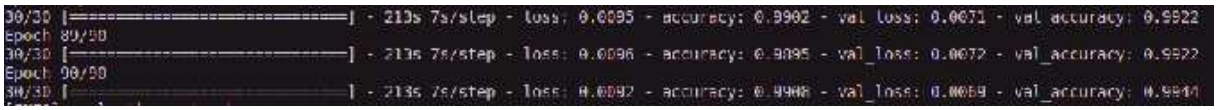
Figura 15 – Exemplos de imagens para cada classe: palma (a-c), punho (d-f), paz (g-i), hangloose (j-l), saudação vulcana (p-r) e "desconhecido"(s-u)



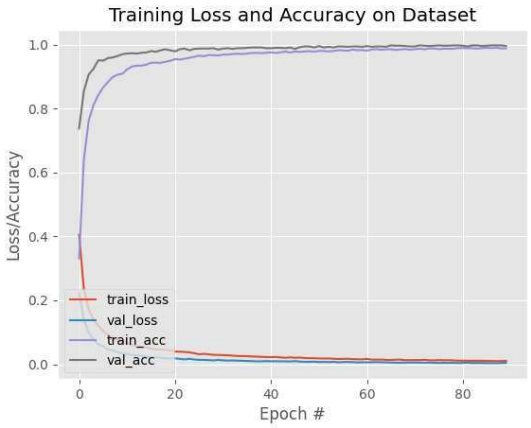
Fonte: Autoria própria

a etapa de *fine-tuning*, quando todo o modelo está descongelado e existe um retreinamento; como o modelo (com exceção de algumas camadas) já havia sido treinado na etapa anterior, essa etapa inicia com valores altos de acurácia. É possível perceber através dos gráficos que o treinamento resultou em um modelo com boa acurácia, pois os valores de acurácia de validação ("val_acc": proporção dos testes de validação com classificações corretas) são próximos de 1.

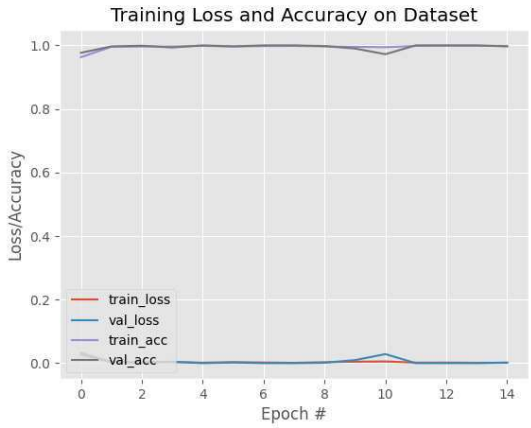
Figura 16 – Métricas e curvas de treinamento do classificador: (a) Valores de treinamento das últimas três épocas, (b) Gráfico de treinamento na etapa de *transfer learning*, (c) Gráfico de treinamento da etapa de *fine tuning*.



(a)



(b)



(c)

Fonte: Autoria própria

As métricas de treinamento e validação podem ser analisadas nos seguintes casos:

- *Overfitting*: a curva de *training loss* tende a 0, porém a curva de *validation loss* tende a um valor bem mais alto. Significa que o modelo aprendeu bem os dados de treinamento, mas não é capaz de generalizar para casos novos.
- *Underfitting*: as curvas de *training loss* e *validation loss* tendem a valores altos. Significa que o modelo não está conseguindo aprender.
- *Fitting* perfeito: as curvas de *training loss* e *validation loss* tendem a valores próximos de zero e, em geral, a curva de *validation loss* fica levemente acima. Significa que o modelo está aprendendo bem e tendo bons resultados para dados novos.

Com a análise das curvas de treinamento, é possível perceber que o modelo treinado conseguiu aprender corretamente e teve uma boa generalização dos dados. Os testes práticos com o classificador demonstraram bons resultados. Porém em casos em que o fundo da imagem possui muitos detalhes, podem ocorrer erros de classificação.

Para chegar ao resultado demonstrado, o conjunto de imagens foi adaptado várias vezes, bem como o treinamento foi realizado com outras arquiteturas de CNN e diversos hiperparâmetros de treinamento para cada uma delas.

Foram testadas outras arquiteturas de CNN: uma versão simplificada da arquitetura VGG16 e também a própria VGG16, cujas métricas de treinamento foram bastante boas, porém com resultados não tão bons para testes com imagens obtidas diretamente da

câmera. Quando a arquitetura ResNet foi testada, os resultados foram muito melhores e foi considerado desnecessário testar outra arquitetura.

Os hiperparâmetros de treinamento adequados são obtidos através de testes e observação dos resultados do treinamento. Aqui serão explicados de forma bastante simplificada a lógica utilizada para isso: para o treinamento foi necessário testar o número de épocas de treinamento, a taxa de aprendizagem e o tamanho do lote de imagens. Com a utilização do algoritmo de otimização *Stochastic Gradient Descent*, a taxa de aprendizagem não pode ser muito grande nem muito pequena: se muito pequena, o modelo pode demorar muito para aprender ou até mesmo não aprender nunca; se muito grande, o modelo tem seus pesos atualizados de maneira errada, levando a comportamentos divergentes. O tamanho do lote de imagens, ou seja, quantas imagens são vistas pelo modelo a cada época, define a variação dos dados: se muito pequena, pode dificultar na generalização dos dados; se muito grande, pode atrapalhar na generalização. O número de épocas define quantas vezes o modelo vai ver uma parte dos dados (lote) e atualizar os pesos; e é definida através da observação de quantas épocas são necessárias para atingir boas métricas de aprendizado.

Apesar de o modelo ter sido treinado para o reconhecimento de seis gestos diferentes, não foi necessária a utilização de todos para a implementação dos comandos. Entre eles foram escolhidos os que apresentaram os melhores resultados em testes práticos.

Na Figura 17 há três exemplos de classificação de gestos. Nela é possível perceber que mesmo com interferência de objetos ao fundo, a classificação é correta.

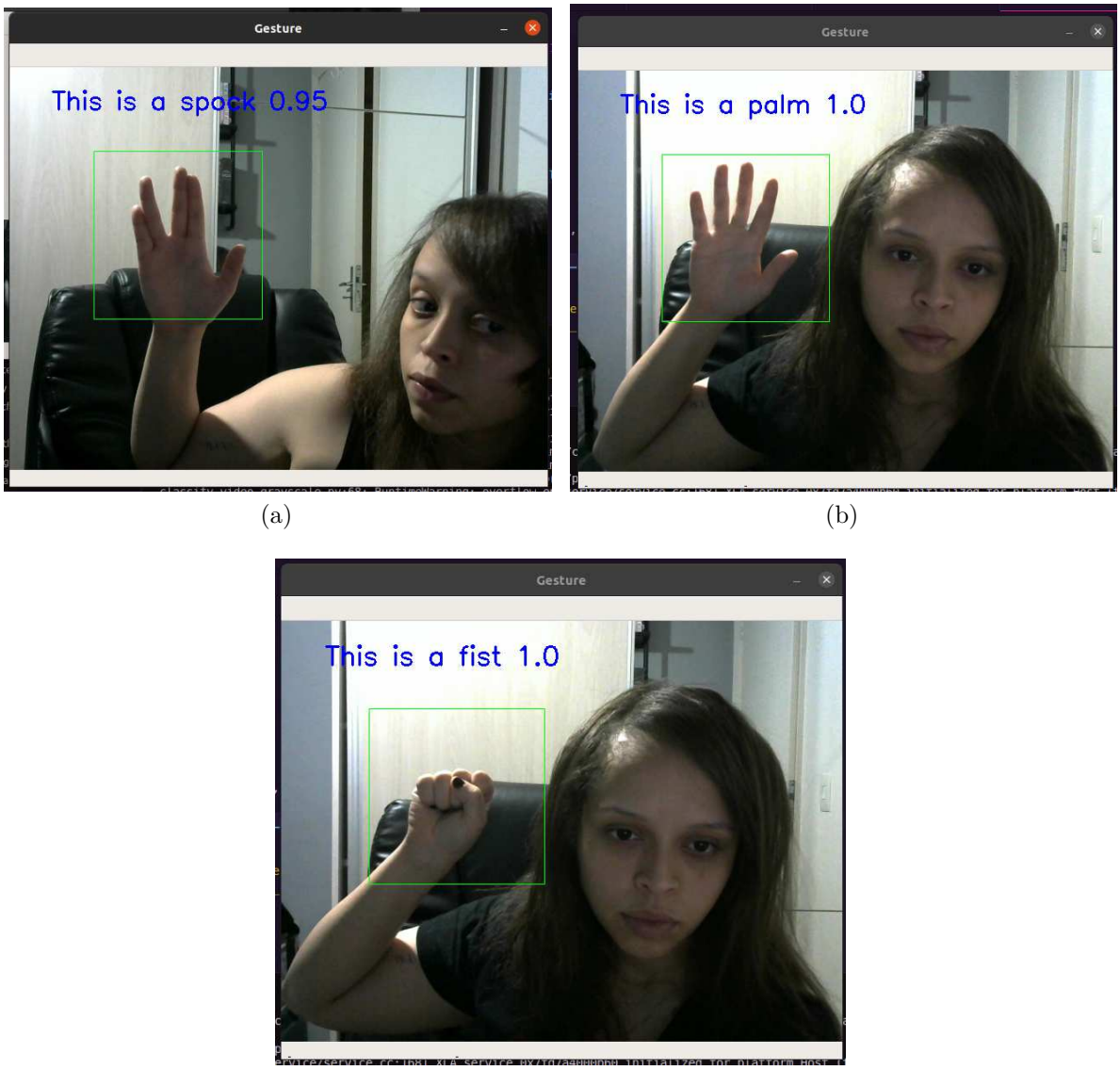
5.8.3 INTEGRAÇÃO DOS MODELOS

Após o treinamento de ambos os modelos, foi escrito o código de reconhecimento de gestos, que integra ambos. O processo, apresentado de forma simplificada no diagrama da Figura 18, consiste em utilizar o detector de mãos para obter a região de interesse (ROI), ou seja, a região da imagem em que se encontra a mão, a partir do *frame* e então submetê-la ao classificador de gestos. Para diferenciar a detecção de movimentação da mão de um gesto estático, utilizou-se uma lógica que verifica se a mão fica parada na imagem por pelo menos dois segundos e só então usa a classificação, caso contrário, detecta o sentido de movimento.

A movimentação da mão é detectada através do cálculo da posição da mão em cada *frame*: a partir da região de interesse, calcula-se as coordenadas médias da mão na imagem, e compara-se a posição calculada com a posição no *frame* anterior; é normal haver uma pequena diferença entre as posições encontradas com esse método, mesmo quando a mão está parada, porém quando a diferença ultrapassa certo valor (determinado através de testes) em algum dos quatro sentidos, considera-se que houve movimento.

Para permitir que o classificador tivesse mais chances de fazer a classificação correta do gesto, era necessário que a região de interesse obtida pelo modelo de detecção de mãos contivesse a imagem completa da mão. Através da observação das imagens da

Figura 17 – Imagens mostrando três casos de classificação de gestos diferentes, com o respectivo valor de certeza do classificador: (a) Spock, (b) "Palm"/Mão aberta e (c) "Fist"/Mão fechada.



(a) (b)

(c)

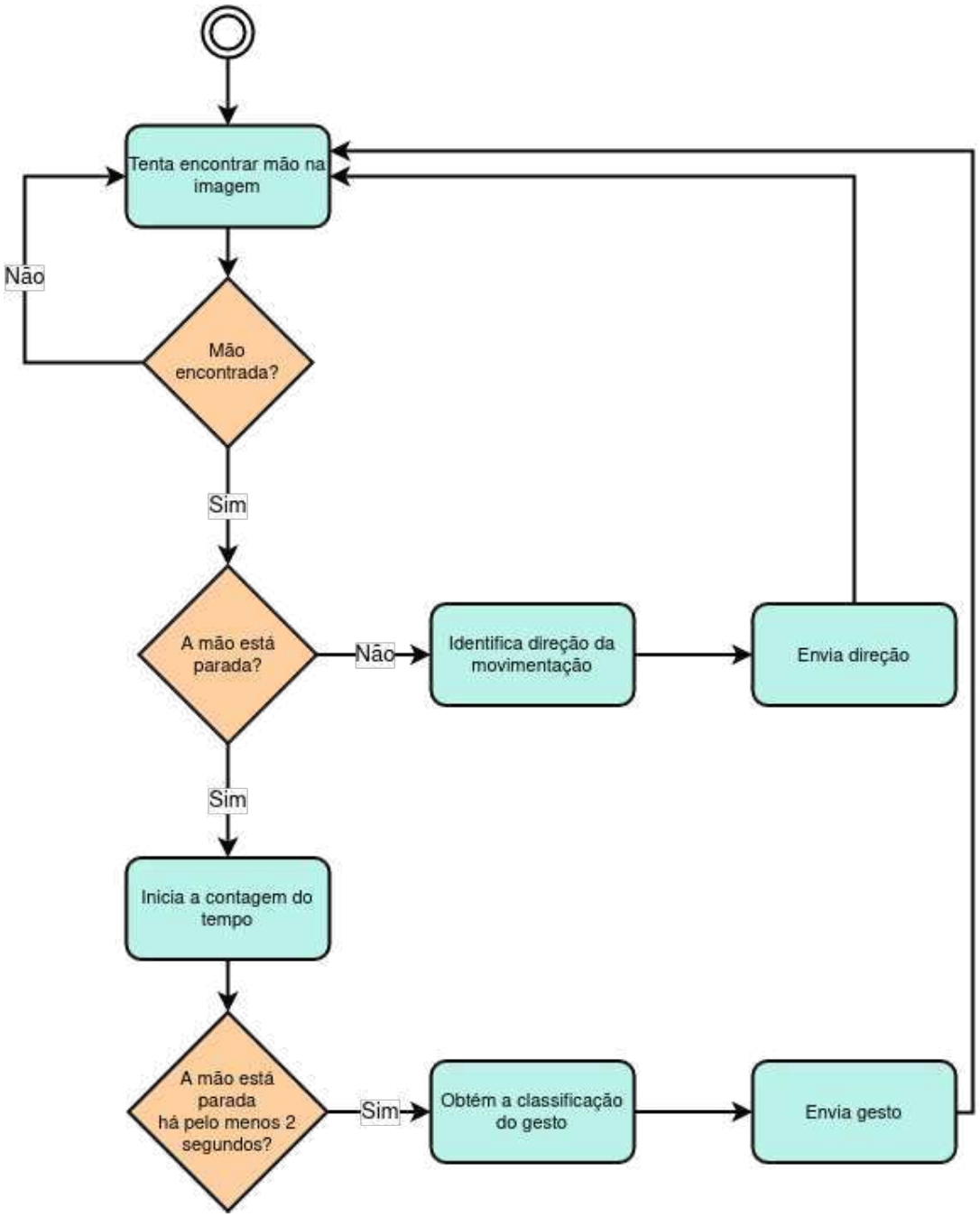
Fonte: Autoria própria

mão obtida pelo detector, foi possível ver que nem sempre isso acontecia, e por isso foi necessário utilizar uma lógica para aumentar um pouco a área obtida, como forma de aumentar as chances de obter a imagem da mão completa. Isso ajudou a aumentar a acurácia da classificação de gestos.

5.9 DESENVOLVIMENTO DO RECONHECIMENTO DE FALA

Como alternativa ao reconhecimento de gestos, foi desenvolvido um sistema de reconhecimento de fala para controle do espelho. O reconhecimento de fala permite que o

Figura 18 – Diagrama do sistema de reconhecimento de gestos.



Fonte: Autoria própria

usuário possa escolher a forma que prefere para interagir com o sistema e também é uma forma de tornar o projeto mais inclusivo.

O reconhecimento de fala foi implementado utilizando-se a linguagem Python, com a biblioteca Speech Recognition. Através dela, o som é capturado por um microfone *usb* conectado ao Raspberry Pi. A identificação de fala é feita com uma função de ajuste do sistema para o ruído do ambiente e com uma função de escuta, que analisa a intensidade do som recebido e passa a capturá-lo quando essa intensidade ultrapassa certo limiar. O

som capturado é então enviado para a API de reconhecimento de fala do Google, a qual o converte para o texto correspondente. O texto é então comparado com um dos possíveis comandos configurados para a interação com o espelho e caso seja válido, é enviado para a interface por uma requisição HTTP. Os comandos válidos são:

- **tarefas**: configura a interface para exibir a lista de tarefas do usuário;
- **eventos**: configura a interface para exibir a lista de eventos;
- **tempo**: configura a interface para exibir as informações meteorológicas;
- **cima**: seleciona o item acima da lista;
- **baixo**: seleciona o item abaixo da lista;
- **apagar**: apaga o item selecionado;
- **voltar**: configura a interface para exibir a tela principal;
- **sair**: encerra a sessão de uso e configura a interface para exibir a tela de despedida.

Comumente, em sistemas de reconhecimento de fala, são configuradas palavras para ativar o funcionamento do sistema ou para sinalizar ao sistema que o trecho de fala seguinte deve ser processado. No caso deste projeto, esse recurso não foi utilizado pois os comandos só serão analisados quando o usuário tiver sido reconhecido pelo espelho.

É válido ressaltar que seria possível, como alternativa, criar um conjunto de dados de voz com os comandos utilizados e treinar um modelo de reconhecimento de fala. Seria possível também utilizar algum modelo já treinado de reconhecimento de fala *offline*. No entanto, essas opções significariam ter mais um processo consumindo bastantes recursos do SBC paralelamente aos modelos de identificação de gestos, o que torna tais opções ruins para este projeto.

Foram feitos testes com alguns modelos de microfone, até chegar em um que fosse capaz de capturar áudio com qualidade necessária para obtenção de bons resultados. É válido ressaltar que a posição do microfone em relação ao usuário também pode ser um fator importante para a qualidade do áudio capturado, porém, como a proposta do espelho envolve a preocupação com a estética, os lugares de posicionamento do microfone na estrutura ficam bastante limitados.

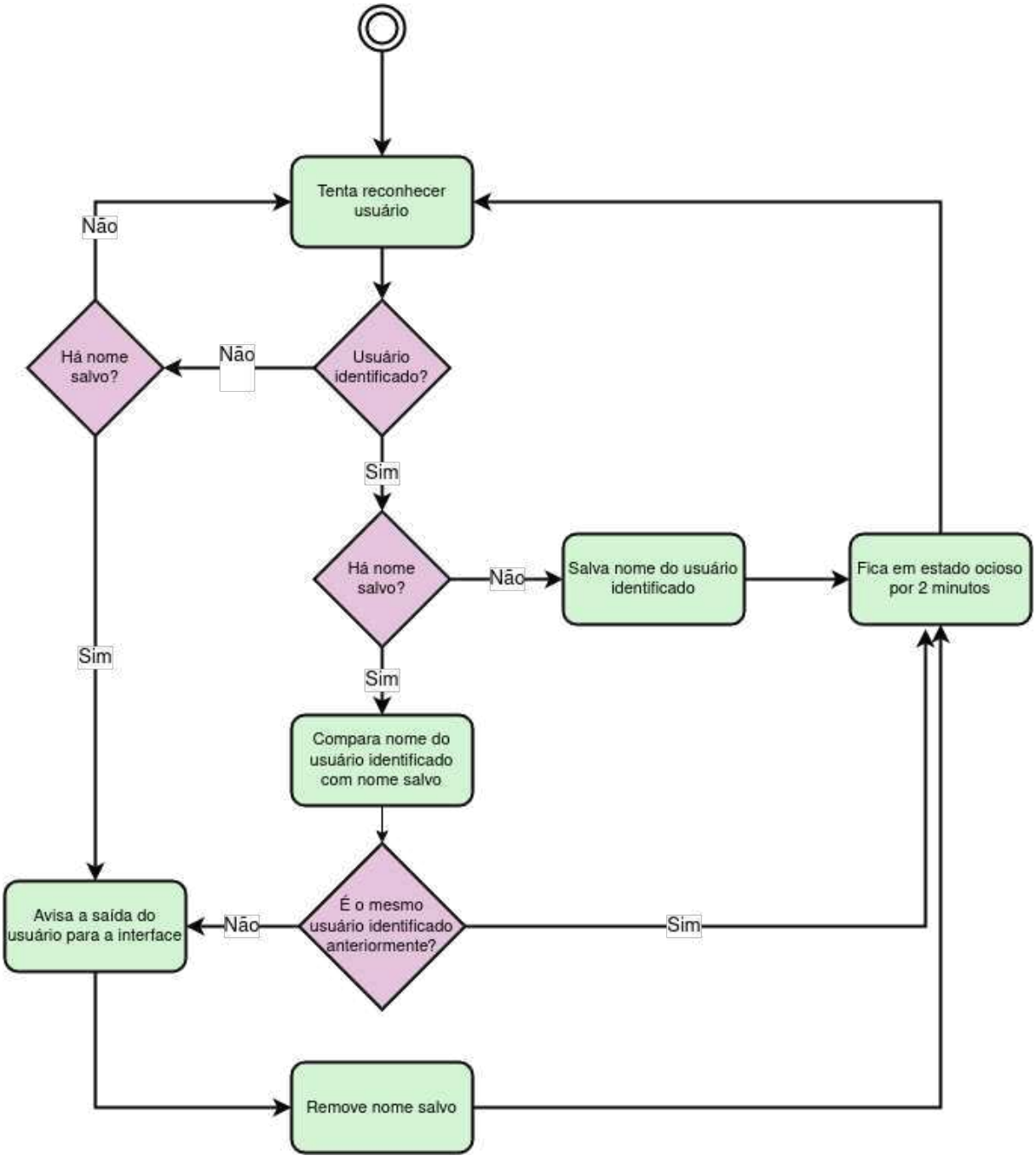
5.10 INTEGRAÇÃO DAS PARTES

A comunicação entre as partes do sistema foi feita toda com base em APIs, utilizando-se requisições HTTP. A aplicação da interface possui uma API com *endpoints* separados para reconhecimento de usuário e saída de usuário, para gestos e direções de movimentação da mão, e para comandos por fala. Os comandos por gestos e fala são aceitos apenas se anteriormente a interface recebeu uma requisição indicando a identificação de um usuário; do mesmo modo, eles não são mais aceitos se for identificada a saída do usuário da frente do espelho, até que se identifique um usuário novamente.

Para que o sistema tenha uso e interação com o usuário mais dinâmicos, foi desenvolvida uma solução para verificar se o usuário saiu da frente do espelho. O sistema

de reconhecimento de faces verifica se a face do usuário é detectada a cada dois minutos, se, no entanto, o usuário não for encontrado, ou um usuário diferente tenha sido encontrado, a interface recebe o comando para esconder as informações, retornando ao estado inicial. A partir de então, o sistema de reconhecimento de faces fica inativo por dois minutos, para então voltar a procurar por um usuário. Esse processo é descrito no diagrama da Figura 19.

Figura 19 – Diagrama do sistema de integração.



Fonte: Autoria própria

Para facilitar a execução paralela dos processos de reconhecimento de gestos e reconhecimento facial, foi acrescentado ao projeto um módulo de câmera para Raspberry Pi, cujo uso não havia sido julgado necessário na análise e modelagem do projeto. O uso do módulo de câmera se mostrou a maneira mais simples de resolver o problema do uso das imagens da câmera por dois processos diferentes, considerando-se que apenas um dos processos poderia ter acesso direto aos *frames* da câmera por vez.

5.11 OTIMIZAÇÃO DO SISTEMA

Os modelos de *Deep Learning* treinados para o sistema de controle por gestos dependem de uma quantidade grande de recursos computacionais, como por exemplo, bastante uso de CPU e memória. Com a utilização do Raspberry Pi, foi possível executar os modelos de detecção e classificação concomitantemente, porém sua capacidade de processamento é bastante limitada e a velocidade de ambos os modelos foi bastante prejudicada, o que tornou o sistema de reconhecimento de gestos muito lento e, por sua vez, tornou o funcionamento do espelho igualmente lento.

Como opções de tornar o sistema mais rápido, havia trocar o Raspberry Pi 4 por uma placa mais potente, como por exemplo uma NVIDIA Jetson Nano; ou encontrar formas de otimizar os modelos. Por falta de acesso a um SBC mais potente, foi necessário fazer a otimização.

O Tensorflow oferece uma solução específica para tornar a execução de modelos treinados mais rápida em dispositivos móveis e de IoT, chamada Tensorflow Lite[13] (TFLite). Com ele é possível obter modelos otimizados e uma melhora significativa no tempo de inferência.

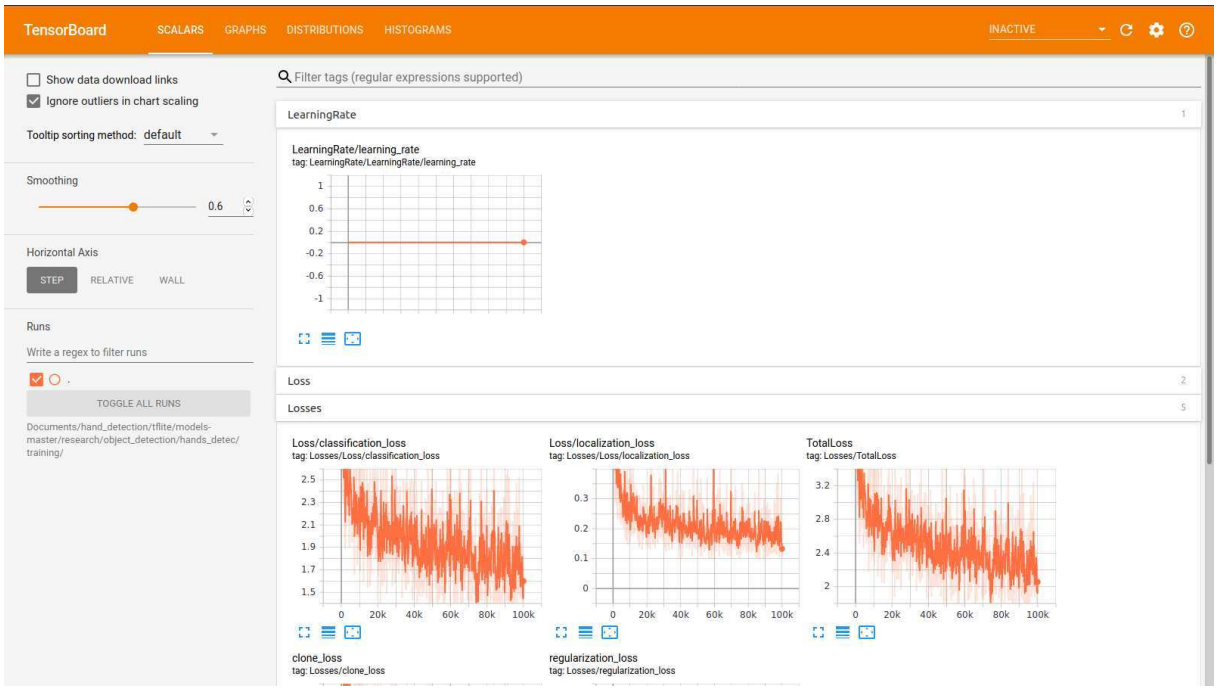
5.11.1 NOVO TREINAMENTO DO MODELOS DE DETECÇÃO E CLASSIFICAÇÃO

Para utilizar o TFLite foi necessário retreinar o modelo de detecção de mãos utilizando o mesmo dataset, pois o modelo treinado anteriormente não permitia a conversão. Dessa vez, porém, foi utilizado o modelo SSD MobileNet V2 Quantized préviamente treinado com o mesmo dataset Coco 2017, o qual, na época do desenvolvimento do projeto, era a melhor opção considerando velocidade e precisão. Dessa vez o modelo foi treinado em 100.000 épocas, também com taxa de aprendizagem de 0.004. Após a conclusão da otimização, o Tensorflow disponibilizou a API de detecção de objetos para a versão 2 do *framework*, que apresenta modelos ainda mais rápidos e precisos.

O modelo demorou cerca de 33 horas para ser treinado e atingiu o valor de perda de validação em torno de 3.2, assim como o modelo anterior. Ele também apresentou bons resultados com testes práticos. A Figura 20 apresenta algumas curvas de treinamento obtidas através da ferramenta Tensorboard.

[13] <https://www.tensorflow.org/lite> (acessado em 30/06/2020)

Figura 20 – Métricas de treinamento.



Fonte: Autoria própria

Foi necessário, também, realizar a conversão do modelo de classificação para um modelo *tflite quantized*, e fazer alguns ajustes no código.

Com a utilização do TFLite, foi possível, aproximadamente, quadruplicar a velocidade de inferência inicial.

5.12 CONSTRUÇÃO DA ESTRUTURA

A estrutura foi feita em madeira e vidro. Para simular o efeito de espelho, foi necessário utilizar *insulfilm*, pois ele permite visualizar a interface exibida no monitor. Todo o *hardware* necessário foi colocado dentro da estrutura. Ela foi desenhada de forma a ser esteticamente agradável e ocultar o *hardware* utilizado.

A figura 21 mostra o resultado final da estrutura.

5.13 TESTES

Nesta Seção são descritos os testes utilizados para realizar a validação do funcionamento das partes do sistema, bem como os resultados e conclusões retiradas desses testes.

Idealmente, os testes seriam feitos com algumas pessoas de diferentes etnias, em diferentes horários e ambientes, para gerar resultados mais próximos ao real do funcionamento do sistema e mais representativos, porém como o trabalho foi feito por

Figura 21 – Estrutura do espelho: (a) parte da frente, (b) parte interior e (c) espelho funcionando.

(a)

(b)

(c)

Fonte: Autoria própria

uma única pessoa, e pela dificuldade de fazer testes com mais pessoas devido à pandemia, os testes foram feitos apenas com a autora.

5.13.1 TESTES DO RECONHECIMENTO DE FALA

Os testes para o reconhecimento de fala visavam abordar dois principais aspectos: precisão do reconhecimento e velocidade. O primeiro está relacionado à necessidade de um usuário ter seus comandos reconhecidos sem precisar de muitas tentativas (repetições do mesmo comando) e o segundo à necessidade do usuário de ver seus comandos terem efeito sobre a interface em pouco tempo.

Eles foram realizados com a repetição dos comandos por dez vezes e observação do número de acertos. Como o reconhecimento é feito através de uma API, para a medição do tempo observou-se os tempos de resposta da API para o cálculo do tempo médio, pois esse

é o tempo que possui mais impacto no tempo percebido pelo usuário. O tempo total, após o usuário falar o comando, equivale à soma do tempo de resposta da API de reconhecimento com o tempo de envio do comando para a interface, que pode ser desconsiderado.

É importante ressaltar que a qualidade do reconhecimento também está diretamente ligada ao volume da voz do usuário e à dicção na pronúncia das palavras; quanto mais clara a pronúncia do comando, melhor é taxa de acerto.

Abaixo encontra-se a Tabela 1 com o número de acertos do sistema de reconhecimento durante o teste, para cada comando, considerando 10 tentativas para cada um deles.

Tabela 1 – Resultado dos testes para reconhecimento de comandos de fala.

Comando	**Número de acertos**
"Tarefas"	8
"Eventos"	9
"Tempo"	10
"Cima"	8
"Baixo"	9
"Apagar"	10
"Voltar"	9
"Sair"	10

A Figura 22 apresenta duas imagens com alguns dos resultados para os testes, em que é possível ver também os tempos aproximados de resposta da API de reconhecimento.

Figura 22 – Imagens mostrando os resultados de testes para reconhecimento de fala: (a) alguns resultados para os comandos "apagar"e "voltar"e (b) alguns resultados para os comandos "eventos"e "tarefas".

(a) (b)

Fonte: Autoria própria

Considerando todos os comandos, a porcentagem média de acerto foi 91,25%. O

tempo médio medido foi de 0,896 segundos. Estes resultados mostram que o sistema de reconhecimento de fala tem bom desempenho e possui velocidade adequada.

Nas imagens da Figura 22 também é possível perceber uma variação grande nos tempos aproximados de resposta da API e isso pode ser devido às oscilações na velocidade da internet durante a realização do teste.

5.13.2 TESTES DO RECONHECIMENTO DE FACE

O reconhecimento de face passou por testes que visavam definir sua precisão, ou seja, definir se o usuário é reconhecido corretamente todas as vezes, e tempo de reconhecimento, ou seja, o tempo necessitado para que o sistema reconheça o usuário após ele se posicionar em frente ao espelho.

Para a realização dos testes, o modelo de reconhecimento foi treinado utilizando imagens da autora e imagens de dois atores, um homem e uma mulher, para fins de avaliação. Foram apresentadas ao sistema dez imagens diferentes das três pessoas, de maneira alternada, e verificado o número de classificações corretas. A Tabela 2 apresenta os resultados obtidos.

Tabela 2 – Resultado dos testes para reconhecimento de usuário.

Pessoa	**Número de acertos**
"Autora"	10
"Atriz"	10
"Ator"	10

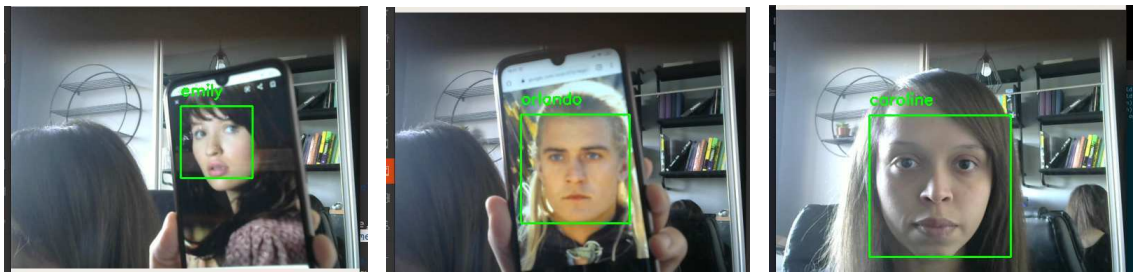
A porcentagem de acerto obtida com os testes foi de 100% e o tempo de reconhecimento foi entre 1 e 2 segundos. Considerando-se que na prática o sistema analisa mais de uma imagem do usuário antes de apresentar o resultado, a porcentagem de acerto e tempo obtidos são bastante bons. No entanto, o mais provável é que esse resultado não represente a acurácia real do sistema, pois não foi possível simular um ambiente real: pessoas reais, em diversos horários do dia e com variações na aparência (acessórios, maquiagem, diferentes penteados, etc); então a acurácia real é provavelmente um pouco menor que a obtida nos testes.

A Figura 23 mostra alguns exemplos de resultados dos testes de reconhecimento de face.

5.13.3 TESTES DO RECONHECIMENTO DE GESTOS

Os testes do sistema de reconhecimento de gestos visavam definir a precisão e a velocidade do reconhecimento. Assim como para o reconhecimento de fala, o aspecto da precisão se relaciona ao fato de que o usuário precisa ter seus comandos reconhecidos sem necessitar de diversas tentativas e que, além disso, os comandos não realizem ações

Figura 23 – Imagens mostrando os resultados de testes para reconhecimento de face: (a) Atriz identificada e (b) ator identificado e (c) autora.



(a) (b) (c)

Fonte: Autoria própria

indesejadas devido a falhas de reconhecimento; e o aspecto da velocidade está relacionado à necessidade de que os comandos tenham efeito em pouco tempo.

No caso do reconhecimento de gestos, dois fatores são importantes para se obter uma boa precisão: a iluminação e o ambiente em frente ao espelho. Ambientes pouco iluminados dificultam a detecção de mãos e tornam a classificação dos gestos menos precisa e no caso da classificação do gesto, o que existe atrás da mão do usuário pode inserir detalhes na imagem que confundem o classificador. Portanto, quanto mais iluminado e menos "poluído"o ambiente em frente ao espelho, melhores são os resultados obtidos para o reconhecimento de gestos.

Os testes foram feitos com a repetição dos gestos e observação do número de acertos. O tempo de reconhecimento acabou sendo definido pela forma como o sistema foi desenvolvido: se o usuário deixar a mão parada por pelo menos dois segundos em frente ao espelho, um gesto será reconhecido entre 2 e 3 segundos e, caso contrário, para casos em que o usuário movimenta a mão (para a esquerda, direta, para cima ou para baixo) em menos de dois segundos, o movimento é reconhecido e enviado para a interface logo após a mão sair da frente do espelho, ou seja, em cerca de 1 segundo.

Como forma de fazer uma melhor avaliação, os testes foram realizados em um ambiente não ideal, ou seja, com alguns objetos de fundo para a imagem e sem qualquer preocupação com a direção da iluminação.

A Tabela 3 abaixo apresenta o número de acertos para cada gesto utilizado no controle da interface do espelho. Foram feitas 10 tentativas para cada um deles.

Considerando todos os comandos, a porcentagem média de acerto foi 82,86%. Estes resultados mostram que o sistema de reconhecimento de gestos tem bom desempenho, considerando-se que é um sistema de visão computacional e que está sujeito a diversos fatores. É importante salientar que, devido à falta de pessoas para testarem o sistema, esses resultados não representam o funcionamento real do sistema para pessoas com outras cores de pele, por exemplo.

Tabela 3 – Resultado dos testes para reconhecimento de gestos.

Gesto	**Número de acertos**
"Direita"	9
"Esquerda"	9
"Cima"	8
"Baixo"	7
"Mão aberta"	9
"Mão fechada"	8
"Saudação Vulcana"	8

6 CONCLUSÃO

O aumento pelo interesse e uso de dispositivos inteligentes em ambiente domiciliar abriu espaço para o surgimento dos *Smart Mirrors*, ou espelhos inteligentes, que integram a um espelho comum uma interface de exibição de informações úteis ao usuário, visando trazer mais praticidade ao dia-a-dia.

Nesse contexto, foi proposto neste projeto o desenvolvimento de um espelho inteligente capaz de reconhecer o usuário e apresentar informações personalizáveis através de um aplicativo móvel. Além disso, o espelho deve poder ser controlado através de gestos e fala. O projeto tem como aspecto principal e mais interessante o uso de Inteligência Artificial para o controle do espelho inteligente desenvolvido. Também trata da execução e otimização de um modelo de Inteligência Artificial em um computador de placa única com recursos bastante limitados.

Durante o desenvolvimento, várias dificuldades foram encontradas. Entre elas podem ser citadas como as principais: a construção de um bom conjunto de dados para treinamento do classificador de gestos; a otimização dos modelos treinados para aumentar a velocidade de inferência. Essas dificuldades exigiram muito tempo em pesquisa e inúmeras tentativas até a obtenção de um bom resultado.

A realização do projeto alcançou os objetivos desejados e permitiu o estudo e grande aprendizado sobre a teoria de *Deep Learning* e sobre a prática do treinamento de modelos. Permitiu também aprendizado sobre Internet das Coisas, Bancos de Dados e comunicação entre processos.

Os resultados obtidos foram satisfatórios, considerando-se que em Visão Computacional sempre existe uma margem de erro, ainda que pequena. Mesmo modelos treinados com as arquiteturas de redes mais precisas e com conjuntos de dados muito grandes não possuem 100% de precisão. As partes do sistema mais passíveis de erro (detecção de gestos, de voz e de usuário) apresentaram bons resultados, conforme discutido na Seção 5.13. O sistema de detecção de gestos foi o que apresentou uma precisão um pouco menor, podendo sofrer interferência de elementos do ambiente nas imagens utilizadas e também dependendo de boa iluminação, mas que ainda assim permite a utilização do espelho sem grandes problemas.

Visando melhorar ainda mais o projeto, seria possível sugerir a implementação de novas funcionalidades para o aplicativo móvel: permitir definir a posição das informações na interface do espelho; permitir a escolha da fonte de notícias; compartilhar as notificações do *smartphone* com o espelho. Outra modificação poderia ser a reescrita do aplicativo móvel com tecnologias multiplataforma, como React Native, assim o aplicativo poderia ser compilado também para IOS ou outros sistemas operacionais móveis.

No caso da interface, mais tipos de informações úteis e funções poderiam ser

adicionadas. Há inúmeras possibilidades para tal, desde integração com sensores ou outros dispositivos inteligentes, até mesmo a implementação de formas de interação envolvendo Realidade Aumentada. Seria possível também criar um sistema em que os usuários pudessem fazer o auto cadastro. Com as modificações citadas e a melhoria dos sistemas de reconhecimento facial e de gestos, e o bons resultados em testes feitos com diversas pessoas, o projeto poderia até mesmo virar um produto.

Finalmente, o desenvolvimento do projeto foi bastante desafiador, complexo e multidisciplinar, e os resultados obtidos foram muito gratificantes e concluíram de uma ótima maneira esse período de aprendizagem.

Referências

ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>. Citado na página 21.

BAHGA, A.; MADISETTI, V. **Internet of Things: A Hands-On Approach**. [S.l.]: Arshdeep Bahga & Vijay Madisetti, 2014. 20-21 p. ISBN 978-0996025515. Citado na página 12.

BENÍTEZ, R. et al. **Inteligencia Artificial Avanzada**. [S.l.]: Editora UOC, 2014. ISBN 9788490643211. Citado na página 17.

BESSERER, D. et al. Fitmirror: A smart mirror for positive affect ineveryday user morning routines. **MA3HMI '16: Proceedings of the Workshop on Multimodal Analyses enabling Artificial Agents in Human-Machine Interaction**, 2016. Citado na página 13.

BUDUMA, N.; LOCASCIO, N. **Fundamentals of Deep Learning**. [S.l.]: O'Reilly Media, Inc., 2017. 1-8 p. ISBN 978-1-491-92561-4. Citado 2 vezes nas páginas 17 e 18.

CHHABRA, N. Comparative analysis of different wireless technologies. 2013. Citado na página 24.

CHOLLET, F. et al. **Keras**. GitHub, 2015. Disponível em: <https://github.com/fchollet/keras>. Citado na página 22.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. [S.l.]: Elsevier Editora Ltda, 2004. 59 p. ISBN 978-85-352-1273-0. Citado na página 23.

ELKHODR, M.; SHAHRESTANI, S.; CHEUNG, H. Emerging wireless technologies in the internet of things: a comparative study. **International Journal of Wireless & Mobile Networks (IJWMN) Vol. 8, No. 5**, 2016. Citado na página 24.

ELMASRI, R.; NAVATHE, S. B. **Sistema de Banco de Dados**. [S.l.]: Pearson Education do Brasil Ltda., 2005. 4 p. ISBN 85-88639-17-3. Citado na página 23.

FITZEK, F. H. P.; CHARAF, H. **Mobile Peer to Peer (P2P): A Tutorial Guide**. [S.l.]: John Wiley & Sons Ltd, 2009. 59 p. ISBN 978-0-470-69992-8. Citado na página 24.

KHAN, S.; RAHMANI, H.; ALI, S. A. **A Guide for Convolutional Neural Networks for Computer Vision**. [S.l.]: Morgan & Claypool, 2018. 1-6 p. ISBN 9781681730219. Citado 4 vezes nas páginas 17, 18, 19 e 20.

MAGRANI, E. **Internet das Coisas**. [S.l.]: FGV Editora, 2018. 19-22 p. ISBN 978-85-225-2006-0. Citado na página 22.

MIKOŁAJCZYK, A.; GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. **2018 International Interdisciplinary PhD Workshop (IIPhDW)**, 2018. Citado na página 20.

OPENCV. **Open Source Computer Vision Library**. 2015. Citado na página 19.

PRATT, L. Y. Discriminability-based transfer between neural networks. 1993. Citado na página 21.

RAHMAN, A. S. M. M. et al. Augmented rendering of makeup features in a smart interactive mirror system fordecision support in cosmetic products selection. **2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications**, 2010. Citado na página 13.

SILVERIO-FERNÁNDEZ, M.; RENUKAPPA, S.; SURESH, S. What is a smart device? - aconceptualisation within the paradigm ofthe internet of things. **Visualization in Engineering**, 2018. Citado 2 vezes nas páginas 12 e 23.

SOUZA, C. R. B. d. et al. Sometimes you need to see through walls — a field study of application programming interfaces. **CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work**, 2004. Citado na página 26.

TSE, D.; VISWANATH, P. **Fundamentals of Wireless Communication**. [S.l.]: John Wiley & Sons, 2005. ISBN 978-0-470-56544-5. Citado na página 23.

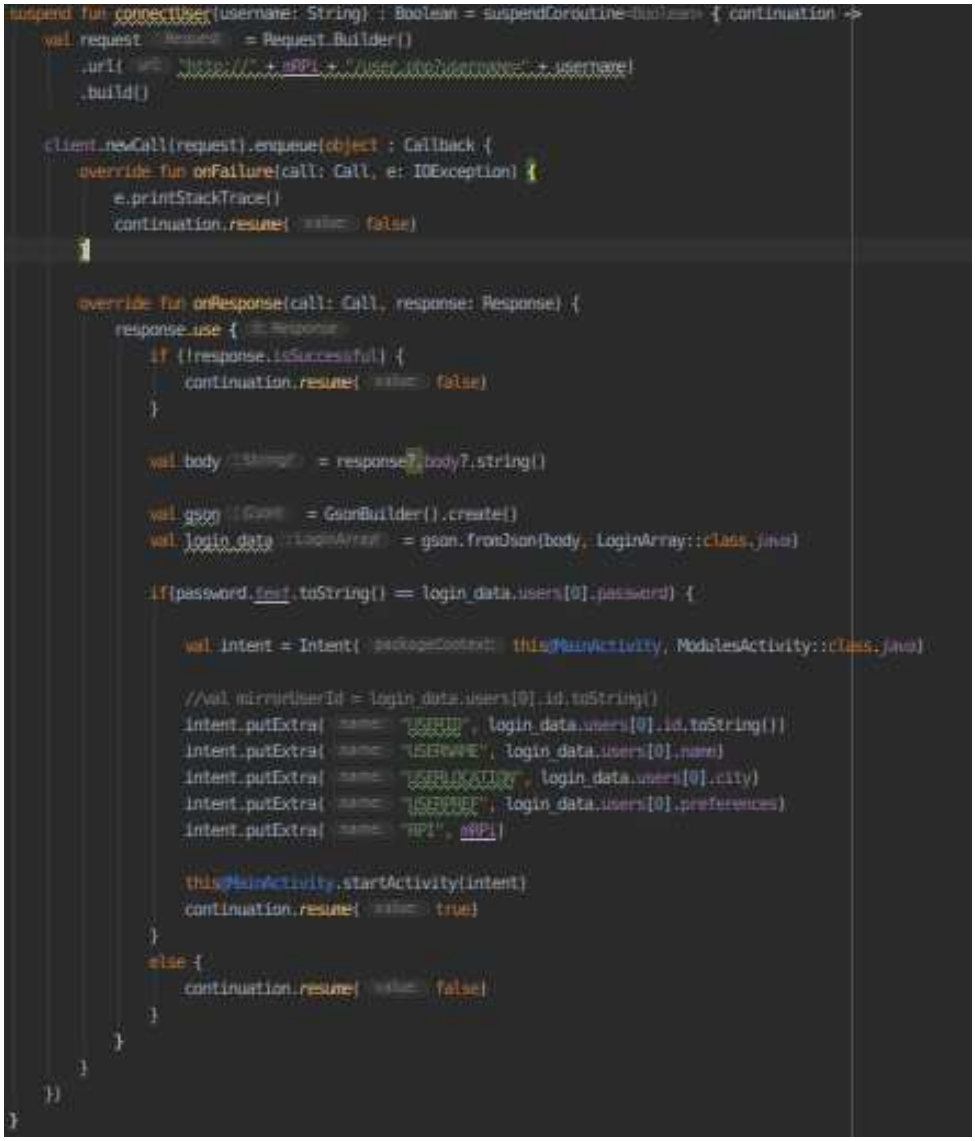
Apêndices

APÊNDICE A – Exemplos do código desenvolvido no projeto

Este capítulo apresenta alguns trechos de código escritos durante o desenvolvimento do projeto.

As Figuras 24, 25, 26 e 27 mostram, respectivamente, as funções de *login*, atualização de configurações do usuário, obtenção e adição de tarefas no aplicativo Android. O código está escrito na linguagem Kotlin.

Figura 24 – Função de *login* do usuário no aplicativo.



Fonte: Autoria própria

As Figuras 28 e 29 mostram as funções responsáveis pelo carregamento em memória dos modelos de inferência de classificação e detecção utilizados no sistema de reconhecimento de gestos. Os trechos de código mostrados foram escritos na linguagem

Figura 25 – Função de atualização de dados do usuário pelo aplicativo.

```
suspend fun updateConfig(userId : String, userName : String, location : String, prefs : String) :
        Boolean = suspendCoroutine<Boolean> { continuation ->

    val formBody :FormBody  = FormBody.Builder()
        .add( name: "userid", userId)
        .add( name: "username", userName)
        .add( name: "city", location)
        .add( name: "preferences", prefs)
        .build()
    val request :Request  = Request.Builder()
        .url( url: "http://" + rpi + "/user.php")
        .post(formBody)
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
            continuation.resume( value: false)
        }

        override fun onResponse(call: Call, response: Response) {
            response.use { it:Response
                if (!response.isSuccessful){
                    continuation.resume( value: false)
                }

                val body :String?  = response?.body?.string()
                continuation.resume( value: true)
            }
        }
    })
}
```

Fonte: Autoria própria

Figura 26 – Função de obtenção da lista de tarefas do usuário.

```
suspend fun getTasks(id: String) : Boolean = suspendCoroutine<Boolean> {continuation ->

    val request :Request  = Request.Builder()
        .url( url: "http://" + rpi + "/get_tasks.php?userid=" + id)
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
            continuation.resume( value: false)
        }

        override fun onResponse(call: Call, response: Response) {
            response.use { it:Response
                if (!response.isSuccessful){
                    continuation.resume( value: false)
                }

                val body :String?  = response?.body?.string()
                val gson :Gson!  = GsonBuilder().create()
                taskList = gson.fromJson(body, TaskArray::class.java).tasks

                runOnUiThread {
                    recyclerToDoView.adapter = MainAdapter( context: this@TasksActivity, taskList)
                }

                continuation.resume( value: true)
            }
        }
    })
}
```

Fonte: Autoria própria

Figura 27 – Função de adição de tarefa usuário.

```
suspend fun insertTask(description: String, userid: Int) : Boolean =  suspendCoroutine<Boolean> {
        continuation ->

    val formBody :FormBody  = FormBody.Builder()
        .add( name: "userid", userid.toString())
        .add( name: "description", description)
        .build()
    val request :Request  = Request.Builder()
        .url( url: "http://" + rpi + "/insert_task.php")
        .post(formBody)
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
            continuation.resume( value: false)
        }

        override fun onResponse(call: Call, response: Response) {
            response.use { it:Response
                if (!response.isSuccessful){
                    continuation.resume( value: false)
                }

                val body :String?  = response?.body?.string()
                println(body)
                isSuccessful = true

                continuation.resume( value: true)
            }
        }
    })
}
```

Fonte: Autoria própria

Python.

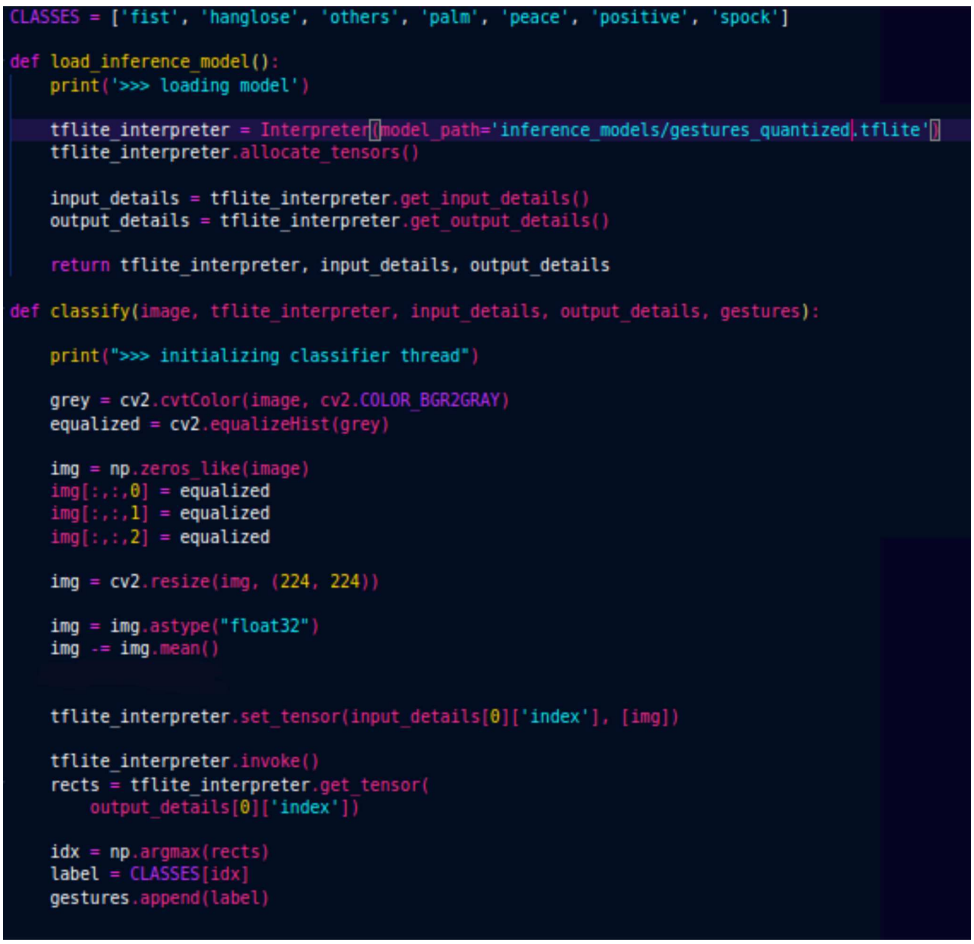
Figura 28 – Função de carregamento do modelo de detecção de mãos e função de detecção.

```
from tflite_runtime.interpreter import Interpreter, load_delegate

CLASSES=['hand']

def load_inference_model():
    print('>>> loading model')

    tflite_interpreter = Interpreter(model_path='inference_models/ssd_mobilenet_v2_quantized.tflite')
    tflite_interpreter.allocate_tensors()

    input_details = tflite_interpreter.get_input_details()
    output_details = tflite_interpreter.get_output_details()

    return tflite_interpreter, input_details, output_details

def detect(image, interpreter, input_details, output_details):

    img = cv2.resize(image, (300, 300))

    interpreter.set_tensor(input_details[0]['index'], [img])
    interpreter.invoke()

    detection_boxes = interpreter.get_tensor(output_details[0]['index'])
    detection_classes = interpreter.get_tensor(output_details[1]['index'])
    detection_scores = interpreter.get_tensor(output_details[2]['index'])
    num_boxes = interpreter.get_tensor(output_details[3]['index'])

    return num_boxes, detection_boxes, detection_scores
```

Fonte: Autoria própria

A Figura 30 apresenta o código de como os as imagens de detecção da mão são

Figura 29 – Função de carregamento do modelo de classificação e função de inferência do classificador.



```
CLASSES = ['fist', 'hanglose', 'others', 'palm', 'peace', 'positive', 'spock']

def load_inference_model():
    print('>>> loading model')

    tflite_interpreter = Interpreter(model_path='inference_models/gestures_quantized.tflite')
    tflite_interpreter.allocate_tensors()

    input_details = tflite_interpreter.get_input_details()
    output_details = tflite_interpreter.get_output_details()

    return tflite_interpreter, input_details, output_details

def classify(image, tflite_interpreter, input_details, output_details, gestures):

    print(">>> initializing classifier thread")

    grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    equalized = cv2.equalizeHist(grey)

    img = np.zeros_like(image)
    img[:,:,0] = equalized
    img[:,:,1] = equalized
    img[:,:,2] = equalized

    img = cv2.resize(img, (224, 224))

    img = img.astype("float32")
    img -= img.mean()


    tflite_interpreter.set_tensor(input_details[0]['index'], [img])

    tflite_interpreter.invoke()
    rects = tflite_interpreter.get_tensor(
        output_details[0]['index'])

    idx = np.argmax(rects)
    label = CLASSES[idx]
    gestures.append(label)
```

Fonte: Autoria própria

utilizadas para o cálculo da direção de movimentação. Já a Figura 31 mostra como é feita a distinção entre a movimentação da mão e um gesto a ser classificado.

A Figura 32 mostra como a direção de movimentação e o gesto identificados são enviados para a interface. Já a Figura 33 mostra como os gestos identificados são recebidos utilizados na interface do espelho.

As Figuras 34, 35, 36 mostram os trechos de código sobre como as informações de tarefas do usuário, notícias e cotação de moedas são obtidas e exibidas na interface.

A Figura 37 mostra um trecho de código do sistema de reconhecimento de fala, escrito na linguagem Python.

Figura 30 – Função de cálculo do sentido de movimentação da mão.

```python
def find_direction(top, bottom, left, right, vertical_pos, horizontal_pos, directions):

    vertical_pos_curr = top + bottom / 2
    horizontal_pos_curr = left + right / 2
    #print('positions: {} {}'.format(horizontal_pos_curr, vertical_pos_curr))

    neutral_direc = False

    vertical_shift = vertical_pos - vertical_pos_curr
    horizontal_shift = horizontal_pos - horizontal_pos_curr

    if (vertical_pos == 0 or horizontal_pos == 0):
        vertical_pos = vertical_pos_curr
        horizontal_pos = horizontal_pos_curr
    elif (vertical_shift > 40):
        directions.append("up")
    elif (vertical_shift < -40):
        directions.append("down")
    elif (horizontal_shift < -40):
        directions.append("right")
    elif (horizontal_shift > 40):
        directions.append("left")
    elif (vertical_shift < 20 and vertical_shift > -20 and horizontal_shift < 20 and horizontal_shift > -20):
        directions.append("neutral")
        neutral_direc = True

    return neutral_direc, vertical_pos_curr, horizontal_pos_curr
```

Fonte: Autoria própria

Figura 31 – Código de distinção entre movimento de mão e gesto.

```python
if (datetime.datetime.now() - firstHandTime).total_seconds() >= 2 and handFound:
    direc = utils.most_frequent(directions)

    if direc == "neutral":
        if((datetime.datetime.now() - lastGestureTime).total_seconds() > 3):
            detected_gesture = utils.most_frequent(gestures)

            x = threading.Thread(target=utils.send_gesture, args=(detected_gesture,))
            x.start()
            lastGestureTime = datetime.datetime.now()

    countingTime = False
    handFound = False
    vertical_pos = 0
    horizontal_pos = 0
    directions = []
    gestures = []

elif (datetime.datetime.now() - lastHandTime).total_seconds() > 0.5 and handFound:
    direc = utils.most_frequent(directions)

    if direc != "neutral" and direc != "":
        if((datetime.datetime.now() - lastDirectionTime).total_seconds() > 2):
            x = threading.Thread(target=utils.send_direction, args=(direc,))
            x.start()

    countingTime = False
    handFound = False
    vertical_pos = 0
    horizontal_pos = 0
    directions = []
    gestures = []
```

Fonte: Autoria própria

Figura 32 – Funções de envio de gesto e direção de movimentação da mão.

```python
def start_user_recognition():
    time.sleep(180)
    try:
        cmd = 'systemctl restart face.service'
        completed = subprocess.run( cmd, shell=True, check=True, stdout=subprocess.PIPE )
    except subprocess.CalledProcessError as e:
        print( 'erro:', e )

def send_gesture(gesture):
    try:
        requests.get("http://localhost:4500/gesture", params={'gesture':gesture})

        if gesture == "spock":
        x = threading.Thread(target=start_user_recognition)
        x.start()

    except Exception as ex:
        print("Could not send gesture: " + str(ex))

def send_direction(direction):
    try:
        requests.get("http://localhost:4500/direction", params={'direction':direction})
    except Exception as ex:
        print("Could not send direction: " + str(ex))
```

Fonte: Autoria própria

Figura 33 – Trecho de código mostrando como os gestos são recebidos e tratados pela interface.

```javascript
socket.on("palm", function(user){

    if(!document.getElementById('todoModal').classList.contains('active') &&
    !document.getElementById('weatherModal').classList.contains('active') &&
    !document.getElementById('countdownModal').classList.contains('active')){

        switch ($('.swiper-slide-active.menu')[0].id){
            case 'todoButton':
                $('#todoModal').addClass('active');
                break;
            case 'weatherButton':
                $('#weatherModal').addClass('active');
                break;
            case 'eventButton':
                $('#countdownModal').addClass('active');
                break;
        }
    }
    else{
        $('.modal').removeClass('active');
        e.preventDefault();
    }
});

socket.on("left", function(user){
    swiperMenu.slidePrev();
});

socket.on("right", function(user){
    swiperMenu.slideNext();
});
```

Fonte: Autoria própria

Figura 34 – Trecho de código mostrando como a lista de tarefas é criada na interface.

```
function updateToDo(){

    todoList = {};

    $.getJSON("todo", function(data) {
        for (i = 0; i < data.length; ++i) {
            todoList[data[i].id] = data[i].description;
        }
        fillList();
    });

}

var fillList = function(){

    listContainer = document.getElementById('todo_swiper');
    listElement = document.createElement('div');
    listElement.setAttribute('id', 'todo_list_swiper');
    listElement.setAttribute('class', 'swiper-wrapper');

    for (var key in todoList){

        listItem = document.createElement('div');
        listItem.setAttribute('class', 'swiper-slide todo');
        listItem.setAttribute('id', 'todo-' + key);
        listItem.innerHTML = '<div class=todoSwiperItem ><span>' + todoList[key] + '</span></div>';

        listElement.appendChild(listItem);
    }

    $('#todo_list_swiper').replaceWith(listElement);

    swiper = new Swiper('.stodo', {
        direction: 'vertical',
        slidesPerView: 4,
        spaceBetween: 30,
        centeredSlides: true,
        observer: true,
        observeParents: true,
        pagination: {
          el: '.swiper-pagination',
          clickable: true,
        },
    });

}
```

Fonte: Autoria própria

Figura 35 – Trecho de código da obtenção e exibição das notícias na interface.

```
var newsList = [];
var itemCounter = 0;

function getNews() {

    $.get('https://cors-anywhere.herokuapp.com/https://feeds.elpais.com/mrss-s/pages/ep/site/brasil.elpais.com/portada', function (data) {

        newsList = [];

        $(data).find("item").each(function () {

            var el = $(this);
            newsList.push([el.find("title").text(), el.find("description").text()]);
        });

    });
};

function showNews() {

    var pos = itemCounter % newsList.length;

    $('#newsTitle').html(newsList[pos][0]);
    $('#newsDescription').html(newsList[pos][1]);

    itemCounter += 1;
};

setInterval(getNews, 1000 * 60 * 5);
getNews();

setInterval(showNews, 1000 * 10);
showNews();
```

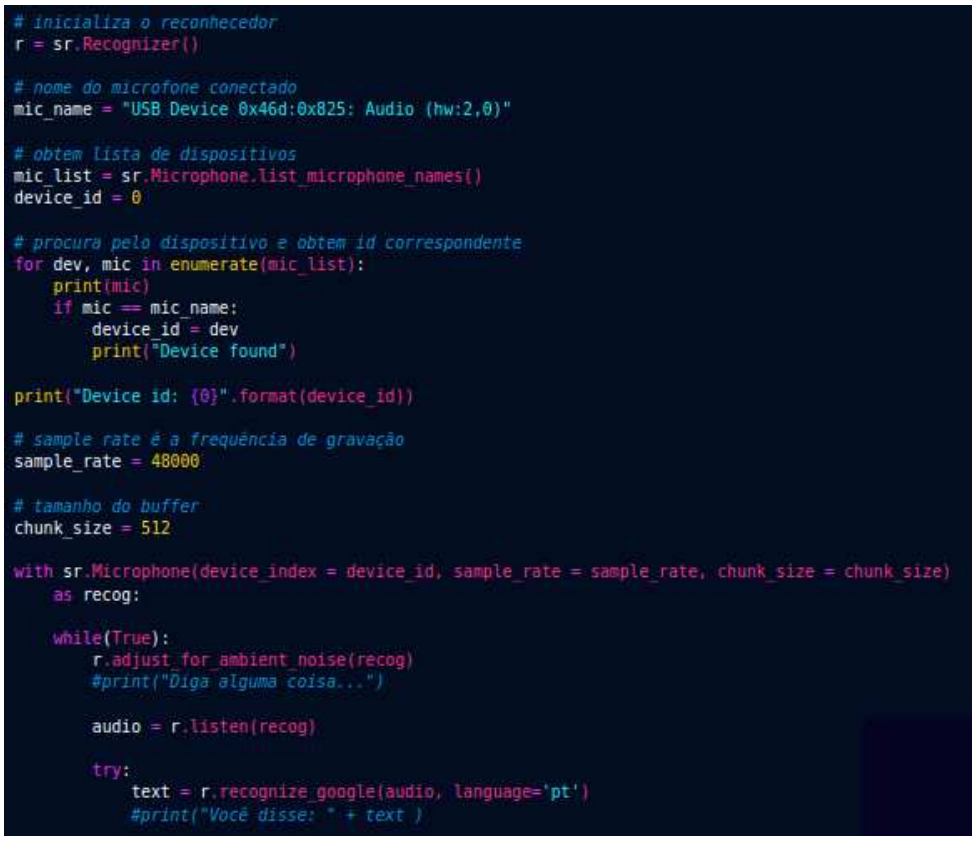
Fonte: Autoria própria

Figura 36 – Trecho de código para obtenção e exibição das cotações de dólar e euro.

```
var getUSDData = function() {

    const Http = new XMLHttpRequest();
    Http.open("GET", 'https://api.exchangeratesapi.io/latest?base=USD');
    Http.send();

    Http.onreadystatechange = function() {
        if(this.readyState==4 && this.status==200) {
            var resp = JSON.parse(this.response);
            $('#usd').html("DÓLAR: " + resp.rates.BRL.toFixed(2) );
        }
    }

    t = setTimeout(function () {
        getUSDData()
    }, 1000 * 60 * 30);
}

var getEURData = function() {

    const Http = new XMLHttpRequest();
    Http.open("GET", 'https://api.exchangeratesapi.io/latest?base=EUR');
    Http.send();

    Http.onreadystatechange = function() {
        if(this.readyState==4 && this.status==200) {
            var resp = JSON.parse(this.response);
            $('#eur').html("EURO: " + resp.rates.BRL.toFixed(2) );
        }
    }

    t = setTimeout(function () {
        getEURData()
    }, 1000 * 60 * 30);
}

getUSDData();
getEURData();
```

Fonte: Autoria própria

Figura 37 – Trecho de código do reconhecimento de fala.



```
# inicializa o reconhecedor
r = sr.Recognizer()

# nome do microfone conectado
mic_name = "USB Device 0x46d:0x825: Audio (hw:2,0)"

# obtem lista de dispositivos
mic_list = sr.Microphone.list_microphone_names()
device_id = 0

# procura pelo dispositivo e obtem id correspondente
for dev, mic in enumerate(mic_list):
    print(mic)
    if mic == mic_name:
        device_id = dev
        print("Device found")

print("Device id: {0}".format(device_id))

# sample rate é a frequência de gravação
sample_rate = 48000

# tamanho do buffer
chunk_size = 512

with sr.Microphone(device_index = device_id, sample_rate = sample_rate, chunk_size = chunk_size)
    as recog:

    while(True):
        r.adjust_for_ambient_noise(recog)
        #print("Diga alguma coisa...")

        audio = r.listen(recog)

        try:
            text = r.recognize_google(audio, language='pt')
            #print("Você disse: " + text )
```

Fonte: Autoria própria