INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIAS E TECNOLOGIA DO AMAZONAS
CAMPUS MANAUS CENTRO

APRENDIZADO INTERATIVO DE INGLÊS BASEADO EM VISÃO COMPUTACIONAL

Gabriely Da Mata Batista

Manaus, Amazonas – Brasil
2023

Biblioteca do *Campus* Manaus Centro - IFAM

B333a Batista, Gabriely Da Mata.
Aprendizado interativo de inglês baseado em visão computacional. / Gabriely Da Mata Batista. – Manaus, 2023.
65 p.: il. color.

Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistema). – Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, *Campus* Manaus Centro, 2023.
Orientador: Prof. Me. Emmerson Santa Rita.

1.Desenvolvimento de sistemas. 2. Aplicativo mobile. 3. Educação infantil 4. Idiomas - Ensino. I. Rita, Emmerson Santa. (Orient.). II. Instituto Federal de Educação, Ciência e Tecnologia do Amazonas. III. Título.

CDD 005.3

Elaborada por Cybelle Taveira Bentes CRB 11/968

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO MÉDIA E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA - AM
DEPARTAMENTO ACADÊMICO DE INFORMAÇÃO E COMUNICAÇÃO
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

TERMO DE APROVAÇÃO

A monografia, que tem como título: **Aprendizado Interativo de Inglês Baseado em Visão Computacional"** foi submetida à defesa pública, sob a avaliação de banca examinadora, como parte dos requisitos necessários para a obtenção do título de graduação do curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

AUTOR (A): GABRIELY DA MATA BATISTA

Monografia aprovada em: 03/07/2023

Orientador (a):

Primeiro (a) examinador (a):

Segundo (a) examinador (a):

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIAS E TECNOLOGIA DO AMAZONAS

CAMPUS MANAUS CENTRO

APRENDIZADO INTERATIVO DE INGLÊS BASEADO EM VISÃO COMPUTACIONAL

Gabriely da Mata Batista

Trabalho de Conclusão de Curso apresentado à banca examinadora Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistema do Instituto Federal de Educação, Ciências e Tecnologia do Amazonas – IFAM Campus Manaus - Centro, como requisito para o cumprimento da disciplina TCC 2– Desenvolvimento de Software

Prof. MsC Emmerson Santa Rita - Orientador

Julho /2023
Manaus, Am

RESUMO

Em vista do contexto atual que vivencia as implicações da globalização, é visível a importância do conhecimento de pelo menos uma língua estrangeira como instrumento de inserção e oportunidades sociais, neste caso o inglês. No Brasil o ensino da língua inglesa é consolidado nos institutos de idiomas e no Ensino Fundamental privado, pois no setor público ainda é deficiente o ensino nas séries iniciais. Por este motivo, faz-se necessário uma forma acessível de auxiliar o aprendizado do inglês nesta faixa etária por meio de um aplicativo *mobile,* desse modo visando solucionar a questão acima, este trabalho propõe utilizar métodos de classificação de imagem para tornar o aprendizado lúdico e interativo. Nesse sentido com a utilização da visão computacional será empregado um modelo de rede neural convolucional para traduzir as imagens em palavras em inglês, assim promovendo um ensino lúdico e acessível.

Palavras-Chave: Educação infantil, ensino de idiomas, inteligência artificial, visão computacional, reconhecimento de padões, aplicativo *mobile*.

ABSTRACT

In view of the current context that experiences the implications of globalization, the importance of knowing at least one foreign language as a tool for insertion and social opportunities is visible, in this case English. In Brazil, the teaching of the English language is consolidated in language institutes and in private elementary education, because in the public sector, teaching in the initial grades is still deficient. For this reason, it is necessary an accessible way to help the learning of English in this age group, in this way, in order to solve the above issue, this work proposes to use image classification methods to make learning fun and interactive. In this sense, with the use of computer vision, a convolutional neural network model will be used to translate the images into words in English, thus promoting a playful and accessible teaching.

Keywords: Early childhood education, language teaching, artificial intelligence, computer vision, pattern recognition, mobile aplication.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

RNA – Rede Neural Artificial

RUP - *Rational Unified Process*

UML - *Unified Modeling Language*

IoT – *Internet of Things*

SSD – *Single Shot Detector*

COCO - *Common Objects in Context*

API - *Application Programming Interface*

URL - *Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

O uso de tecnologias no ensino já é uma realidade, principalmente diante ao crescente número de estudos voltados a novas abordagens de ensino e aprendizagem de línguas. As escolas particulares, com ênfase as de idiomas oferecem novas formas de ensino lúdico e interativo por meio do uso de tecnologias. Entretanto, poucas oferecem uma experiência de usuário por meio de uma visão computacional.

A visão computacional no âmbito da inteligência artificial é ampla. São várias áreas de atuação que possuem soluções através dessa tecnologia, mas há poucas iniciativas em algumas delas, como a área de ensino de idiomas em que grande parte das ferramentas de aprendizagem é feita somente por jogos educacionais que, segundo Dondi e Moretti (2007), são metodologias de ensino com o objetivo didático explícito que podem ser melhorados e adaptados para promover um meio de aprendizado inserido em um contexto formal e informal.

Nesse contexto, o projeto desenvolvido é um aplicativo *mobile* para auxiliar o aprendizado de palavras em inglês de forma interativa com o meio em que vivem, capturando os elementos através da câmera do celular e como retorno sua forma de escrita em inglês e sua pronúncia.

1.1 Problematização

Visto que o conhecimento do idioma inglês não é mais apenas um diferencial e sim uma necessidade no cenário atual, devido às implicações da globalização, segundo Crystal (2003), percebe-se que o ensino convencional muitas vezes não consegue fornecer uma abordagem acessível, lúdica e interativa para auxiliar o aprendizado do inglês nessa faixa etária, com base no que Vygotsky (1998) afirma, atividades que interagem com o ambiente que a criança vive são estimuladores da cognição e criatividade. Nesse sentido, a brincadeira com a intenção de ensinar não é uma atividade ociosa, mas desempenha um papel significativo no processo de socialização do indivíduo. Diante dessa problemática, surge a seguinte questão: como auxiliar o aprendizado das crianças de forma lúdica, levando em consideração a importância do idioma inglês e as limitações do ensino tradicional?

1.2 Justificativa

A utilização de um aplicativo *mobile* se mostra uma escolha relevante no contexto em que os avanços tecnológicos e a popularidade dos dispositivos móveis estão cada vez mais presentes no cotidiano das crianças, tornando-se uma ferramenta familiar e atraente para elas. Ao adotar um aplicativo *mobile*, é possível explorar essa familiaridade e engajamento, pois podem aprender as palavras em inglês em qualquer ambiente em que estiverem, somente apontando a câmera do celular, assim criando uma experiência de aprendizado envolvente e motivadora.

É importante salientar que além do ensino de inglês, a aplicação possui um potencial significativo para ser aplicado em outras áreas do conhecimento, por exemplo na área da Geografia, os alunos poderiam explorar em uma visita técnica e aprender sobre a fauna e flora, apontando a câmera do celular ou tirando uma foto das flores, plantas, animais e o aplicativo reconhecendo cada espécie, desse modo promovendo uma compreensão mais profunda da disciplina.

Esse é apenas um exemplo da utilização de aplicativos *mobile* com recurso de visão computacional, eles podem enriquecer o processo educacional, estimulando a curiosidade, a autonomia e o desenvolvimento das habilidades dos alunos.

1.3 OBJETIVOS

1.3.1 Objetivo geral

Desenvolver um aplicativo *mobile* para crianças de 5 a 10 anos, que utilize visão computacional para promover o aprendizado de palavras em inglês por meio da interação com o ambiente.

1.3.2 Objetivos específicos

Para produzir da melhor forma esse objetivo geral, traçou-se os seguintes objetivos específicos:

- Identificar os métodos mais adequados para o ensino de inglês para o público-alvo deste trabalho
- Selecionar as imagens de objetos baseadas nos métodos de ensino previamente identificados
- Preparar e testar os modelos de visão computacional
- Desenvolver o aplicativo *mobile* integrando os modelos já testados
- Testar o aplicativo utilizando a câmera do celular

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo geral deste capítulo é discutir o embasamento teórico que apoia o tema – Aprendizado interativo de inglês baseado em visão computacional.

2.1 Visão Computacional

De acordo com *Bradski* e *Kaehler* (2008), visão computacional é transformar dados não estruturados como imagens e vídeos em uma nova informação ou promover uma capacidade de decisão. Um dos principais componentes do corpo humano é o olho, capaz de distinguir até 10 mil cores, assim é natural a capacidade de reconhecer e entender uma imagem ou vídeo, já para um sistema computacional que interpreta apenas números, é mais complicado. Segundo *Prince* (2012), em várias décadas estão sendo feitos sistemas de visão, mas até o momento apenas sistemas que são focados em um objetivo específico foram desenvolvidos com sucesso.

A área de visão computacional sofreu um grande avanço nos últimos anos em virtude da utilização de técnicas de inteligência artificial, como as redes neurais, nos processos de extração de características, detecção, identificação e classificação de imagens, entre outros. As redes neurais têm sido amplamente aplicadas em problemas de visão computacional, permitindo que os sistemas computacionais reconheçam objetos, rostos, padrões e realizem tarefas complexas de análise visual. Essa abordagem baseada em redes neurais tem se mostrado eficaz na interpretação e compreensão de imagens, contribuindo para o desenvolvimento de soluções mais robustas e precisas na área de visão computacional.

2.2 Redes Neurais Artificiais

Segundo *Caccia* (2018), as RNAs são abstrações matemáticas que se inspiram no funcionamento do córtex cerebral humano. Seu objetivo é proporcionar um modelo de aprendizado computacional baseado nas mesmas estruturas e processos presentes no cérebro humano. Essas redes são compostas por neurônios artificiais interconectados, capazes de aprender e

generalizar a partir dos dados de entrada, permitindo o processamento de informações complexas e a realização de tarefas.

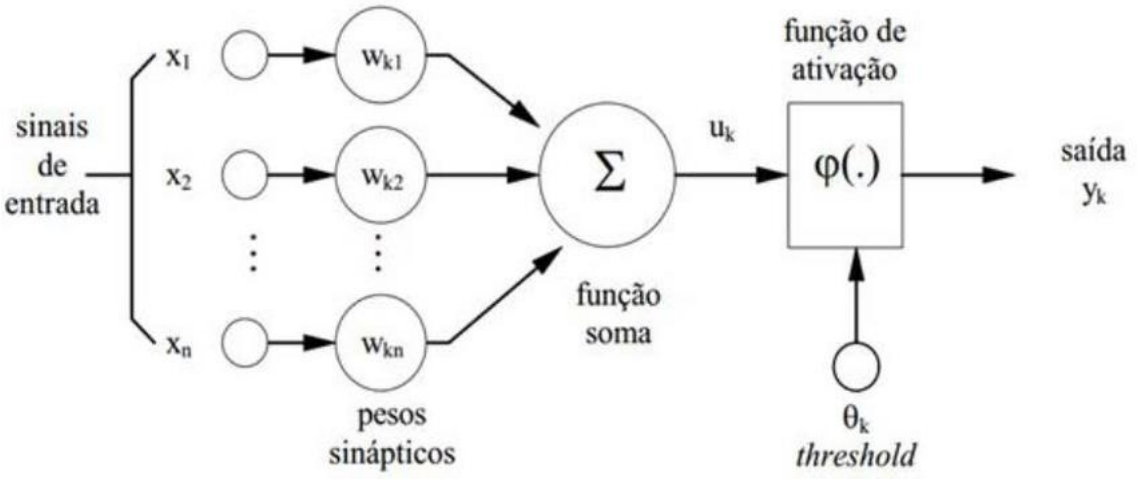
2.2.1 Perceptron

Conforme descrito por *Berger* (2016), o *perceptron* é a forma mais simples de um neurônio artificial. Sua estrutura, representada na Figura 1, é composta por terminais de entrada (dendritos), terminais de saída (axônios), pesos sinápticos associados a cada terminal e uma função de ativação (*threshold*). Essa estrutura é semelhante à estrutura de um neurônio biológico.

No *perceptron*, os terminais de entrada recebem os sinais de entrada, que são multiplicados pelos pesos correspondentes. Esses valores multiplicados são somados e, em seguida, passados para a função de ativação. A função de ativação determina se o neurônio será ativado ou não, influenciando na geração do sinal de saída.

Em resumo, o *perceptron* pode ser visualizado como uma representação matemática simplificada de um neurônio biológico. Ele realiza a combinação linear dos sinais de entrada ponderados pelos pesos sinápticos e, em seguida, aplica uma função de ativação para produzir um sinal de saída. Essa estrutura básica do *perceptron* serve como base para a construção de redes neurais mais complexas.

Figura 1 – Estrutura *perceptron*



Fonte: (Assis, 2016)

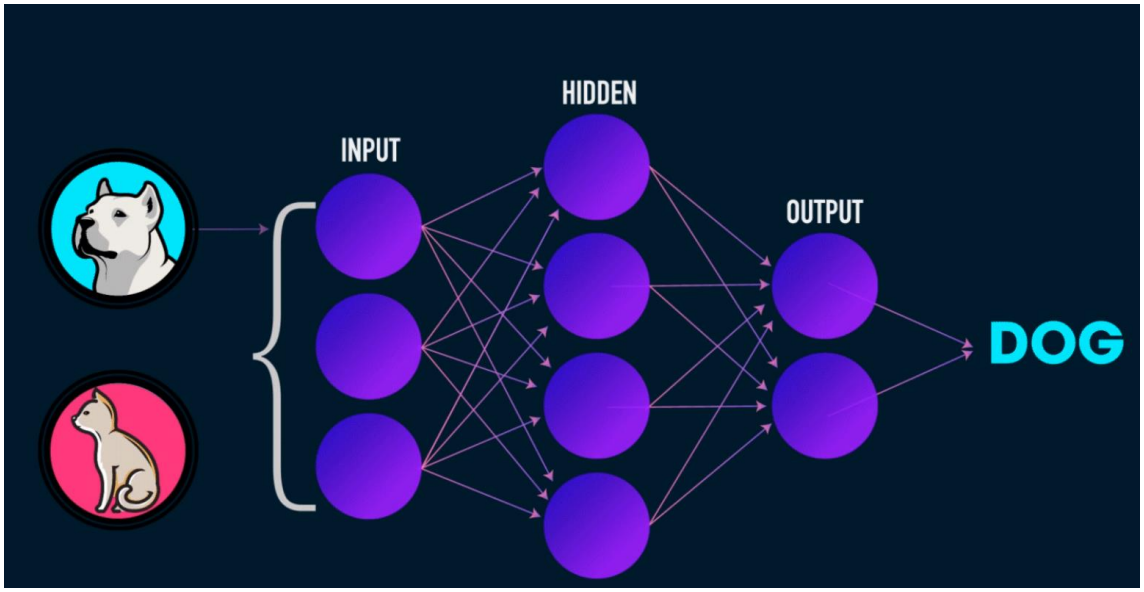
2.2.2 Redes Neurais Convolucionais

O *Perceptron* permitiu uma compreensão clara de como funciona uma rede neural em termos matemáticos (*Data Science Academy*, 2022). Hoje, é mais comum usar outros modelos, como as Redes neurais convolucionais que se destacaram por sua eficiência no reconhecimento e classificação de imagens.

De acordo com Geitgey (2016), as redes neurais convolucionais possuem a capacidade de reduzir a complexidade no tratamento de imagens por meio do uso de camadas de convolução e filtros(*kernels*). Essas camadas simplificam a imagem ao realizar operações de convolução e agrupamento, o que permite uma representação mais compacta e significativa dos dados visuais.

As camadas subsequentes, como as camadas de agrupamento (*pooling*), reduzem a dimensionalidade e agregam informações importantes, segundo Pattanayak (2017), os filtros e as camadas de agrupamento são as camadas ocultas, como mostrado na Figura 2. É importante também que ao utilizar uma camada convolucional ficar atento aos hiperparâmetros a ela atribuídos. A escolha correta deles pode definir o sucesso ou não da aplicação.

Figura 2 - Rede Neural Convolucional



Fonte: (Jobu, 2023)

2.2.2.1 Hiperparâmetros

Como dito anteriormente, os hiperparâmetros são parâmetros ajustáveis que afetam o comportamento e o desempenho de um modelo de rede neural. Eles não são aprendidos diretamente pelos algoritmos de treinamento, mas são definidos pelo desenvolvedor antes do treinamento iniciar. Abaixo estão alguns que serão utilizados ao longo da trabalho.

- Épocas (*epochs*): Uma época significa que cada amostra no conjunto de dados de treinamento foi alimentada pelo modelo de treinamento pelo menos uma vez. Se suas épocas estiverem definidas como 50, por exemplo, isso significa que o modelo que você está treinando funcionará em todo o conjunto de dados de treinamento 50 vezes (*Teachable Machine*, 2018).
- *Batch size*: É um lote, ou seja, o conjunto de amostras usadas em uma iteração de treinamento. Por exemplo, digamos que você tenha 80 imagens e escolha um tamanho de lote de 16. Isso significa que os dados serão divididos em 80/16 = 5 lotes. Depois que todos os 5 lotes tiverem sido alimentados no modelo, exatamente uma época será concluída (*Teachable Machine*, 2018).
- *Learning rate*: A taxa de aprendizado (*learning rate*) determina quanto o valor de cada peso altera a cada iteração e reflete na velocidade com a qual a rede altera sua estrutura de pesos (MAZUR, 2015).

2.2.3 Modelos pré-treinados

Conforme apresentado, as Redes Neurais Convolucionais são uma ferramenta poderosa, principalmente aplicada na visão computacional, como consequência há modelos pré-treinados disponíveis, que permitem o uso de um modelo previamente treinado em problemas semelhantes. Essa abordagem oferece uma maneira mais rápida e pragmática de construir um modelo baseado em Redes Neurais Convolucionais, pois permite pular etapas mais complexas de treinamento e obter resultados excelentes (TUGRUL; ELFATIMI; ERYIGIT,

2022). Desse modo, utilizar um modelo pré-treinado implica aproveitar os conhecimentos adquiridos durante o treinamento em grandes conjuntos de dados e aplicá-los a um problema específico. Dessa forma, é possível aproveitar os recursos de extração de características aprendidos previamente, acelerando o processo de construção do modelo e ainda sim obter resultados eficazes.

2.3 Aprendizado interativo na infância

Como defende Vygotsky (1998) em suas pesquisas, diz-se que a melhor forma de ensino para a criança é ser exposta a um estímulo de atividades cognitivas no ambiente em que está inserida, ou seja, ele pressupõe uma perspectiva do desenvolvimento humano é natural e fluido o pensamento sendo constituído com base no ambiente histórico e cultural: a criança reconstrói internamente uma atividade externa, como resultado de processos interativos que se dão ao longo do tempo. Esta reconstrução é dita por Vygotsky (1998) na lei que denominou de dupla estimulação: tudo que o indivíduo transmite, principalmente nos primeiros anos de vida são situações, palavras que o mesmo aprendeu no meio social desse modo, quando é aprendido e modificado pelo indivíduo e devolvido para a sociedade passa a desenvolver o conhecimento. Assim a criança vai aprendendo e se modificando. Vygotsky (1998) salienta que as possibilidades que o ambiente proporciona ao indivíduo são fundamentais para que este se constitua como sujeito lúcido e consciente, capaz de tomar decisões na sua vida. Diante disso, o acesso a instrumentos físicos ou simbólicos desenvolvidos para o ensino é fundamental.

2.4 Aplicativos Móveis

De acordo com dados das empresas *Sensor Tower* e *AppleInsider*, os aplicativos móveis têm se destacado como uma revolução tecnológica de grande impacto social, visto que esses dispositivos têm se tornado acessíveis à população em geral e disponibilizam uma ampla variedade de softwares em suas lojas. No período analisado, as lojas da *Apple* e do *Google* registraram um total de 8,2 bilhões e 28,3 bilhões de downloads de aplicativos, respectivamente (*Eishima*, 2020). Comparativamente, no mesmo trimestre de 2019, foram realizados 8 bilhões de downloads na loja da *Apple* e 21,6 bilhões na loja do

Google, (*Eishima*, 2020). Essa popularidade e sucesso dos aplicativos móveis se devem, em grande parte, à facilidade de acesso e disseminação dos smartphones. Esses dispositivos portáteis proporcionam mobilidade aos usuários, funcionando como computadores de bolso que podem ser levados consigo a qualquer lugar durante as 24 horas do dia. Além disso, os smartphones são equipados com recursos variados, como câmera digital, GPS, conexão *wireless* e acesso à Internet 3G, 4G e 5G tornando-se uma ferramenta versátil quando combinada com um aplicativo apropriado.

Portanto, a popularização dos aplicativos móveis está intrinsecamente ligada à mobilidade e às características multifuncionais dos smartphones, que proporcionam um acesso fácil e constante aos usuários, conferindo a esses dispositivos um papel fundamental em diversas áreas da vida cotidiana.

2.4 Trabalhos Correlatos

Neste tópico, serão apresentados alguns trabalhos correlatos que abordam aspectos relacionados ao tema do aprendizado interativo de inglês baseado em visão computacional. Os trabalhos selecionados se dividem em duas categorias, a primeira visando fornecer uma visão abrangente das tecnologias interativas de ensino de inglês e a segunda das aplicações que utilizam tecnologias de classificação de imagens.

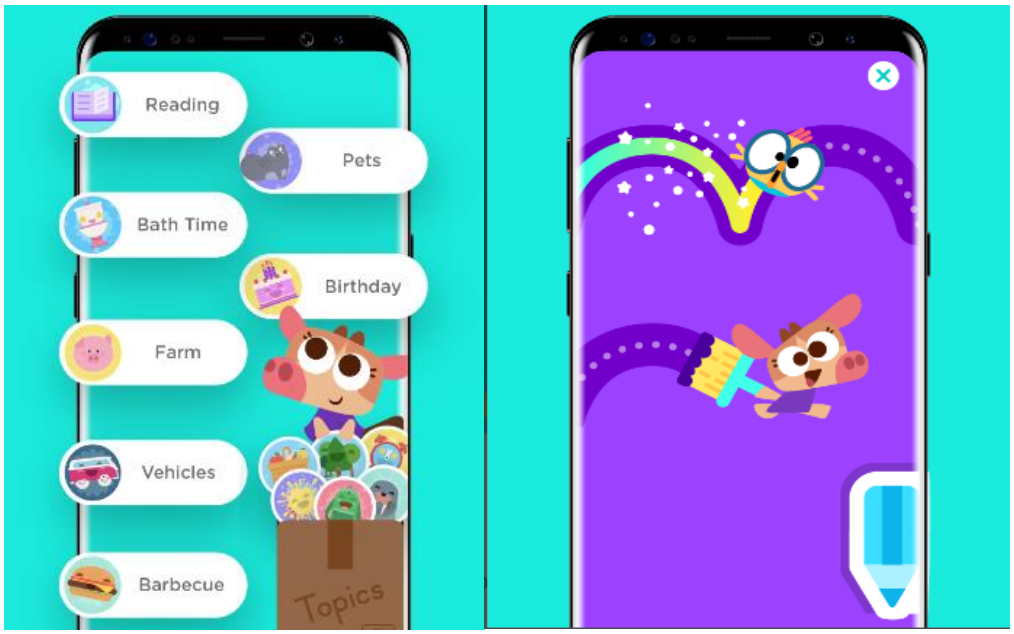
2.4.1 Trabalhos com tecnologias interativas de ensino de inglês

2.4.1.1 *Lingokids*

O primeiro trabalho a ser abordado é o *Lingokids*, um aplicativo móvel desenvolvido em 2015, que oferece uma experiência interativa de aprendizado de inglês para crianças na faixa etária de 2 a 8 anos. Ele foi criado com o objetivo de suprir a carência de aplicativos que proporcionassem um ambiente de aprendizado de inglês adequado às necessidades das crianças, utilizando jogos, histórias e atividades interativas, ou seja, oferece uma abordagem inovadora ao ensino tradicional, proporcionando uma experiência divertida e envolvente para o desenvolvimento das habilidades linguísticas das crianças.

A aplicação do *Lingokids* é desenvolvida em *React Native* e conta com mais de 750 jogos variados, que abrangem desde o aprendizado do alfabeto (ABC) até a prática de pronúncia, vocabulário e gramática. Abaixo seguem as imagens do *app*, contendo os tópicos de ensino que a aplicação aborda e um jogo que estimula a criatividade.

Figura 3 - Telas *Lingokids*



Fonte: Google Play Store LingoKids (2022)

Na Tabela 1 abaixo são apresentadas as principais diferenças e semelhanças entre o aplicativo apresentado ao longo do trabalho (*EnglishAI*) e o *Lingokids*.

Tabela 1 - Diferenças e semelhanças *LingoKids*

	EnglishAI	***Lingokids***
Aprendizado Interativo	Utiliza aprendizado interativo	Oferece uma experiência interativa de aprendizado
Visão Computacional	Utiliza visão computacional	Não utiliza visão computacional
Funcionalidade de Tirar Fotos	Permite que as crianças tirem fotos dos objetos	Não possui funcionalidade de tirar fotos
Aprendizado de Palavras	Fornece palavras em inglês através da visão	Oferece prática de vocabulário em inglês
Pronúncia das Palavras	Inclui a pronúncia das palavras em inglês	Possui recursos de pronúncia e compreensão auditiva

Faixa Etária	5-10 anos	2-8 anos
Disponibilidade de Jogos	Não possui uma variedade de jogos	Oferece mais de 750 jogos e atividades interativas
Desenvolvimento	*Flutter e Dart*	*React Native*

Fonte: Própria autora

Ambos os aplicativos visam tornar o aprendizado de inglês mais envolvente e lúdico, estimulando o desenvolvimento das habilidades linguísticas das crianças. Enquanto o trabalho apresentado neste trabalho (*EnglishAI*) se destaca pela utilização da visão computacional e a interação com o ambiente físico das crianças, o *Lingokids* oferece uma abordagem baseada em jogos e atividades interativas para promover a prática de vocabulário, pronúncia e compreensão do idioma.

2.4.1.2 *Buddy.ai*

O segundo trabalho a ser explorado é o *Buddy.ai*, uma plataforma paga de aprendizado de inglês que também utiliza inteligência artificial para auxiliar as crianças na prática da fala e compreensão auditiva. Ele fornece feedback em tempo real, permitindo que as crianças pratiquem a pronúncia correta e melhorem suas habilidades de comunicação oral. Com uma abordagem mais personalizada e interativa, o *Buddy.ai* oferece um ambiente de aprendizado individualizado, adaptando-se às necessidades e ritmo de cada criança. Abaixo seguem as imagens do aplicativo presentes na *Play Store.*

Figura 4 - Telas *Buddy.ai*

Fonte: *Google Play Store Buddy.ai* (2023)

Na Tabela 2 abaixo são apresentadas as principais diferenças entre o aplicativo apresentado ao longo do trabalho (*EnglishAI*) e o *Buddy.ai*.

Tabela 2 - Diferenças e semelhanças *Buddy.ai*

	EnglishAI	***Buddy.ai***
Aprendizado Interativo	Utiliza aprendizado interativo	Utiliza aprendizado interativo
Visão Computacional	Utiliza visão computacional	Não utiliza visão computacional
Funcionalidade de Tirar Fotos	Permite que as crianças tirem fotos dos objetos	Não possui funcionalidade de tirar fotos
Aprendizado de Palavras	Fornece palavras em inglês através da visão	Oferece prática de vocabulário em inglês
Pronúncia das Palavras	Inclui a pronúncia das palavras em inglês	Oferece recursos de pronúncia e compreensão auditiva, além de reconhecimento de voz
Faixa Etária	5-10 anos	Não especificado
Disponibilidade de Jogos	Não possui uma variedade de jogos	Possui uma variedade de jogos
Preço	Gratuito	Pago
Personalização	Não oferece	Oferece um ambiente de aprendizado individualizado

Fonte: Própria autora

Essas diferenças mostram que cada aplicativo possui abordagens distintas para o ensino de inglês para crianças. Enquanto o aplicativo apresentado ao longo do trabalho (*EnglishAI*) enfatiza a interação com o ambiente físico e a visão computacional, o *Buddy.ai* foca na prática da fala e compreensão auditiva através da inteligência artificial.

Esses trabalhos representam exemplos de como as tecnologias interativas têm sido aplicadas de forma inovadora no ensino de inglês para crianças. Ao combinar elementos lúdicos, interatividade e recursos avançados, essas abordagens oferecem uma nova perspectiva no processo de aprendizagem, buscando tornar o ensino de inglês mais acessível, atraente e eficiente para as crianças.

2.4.2 Trabalho com classificação de imagens

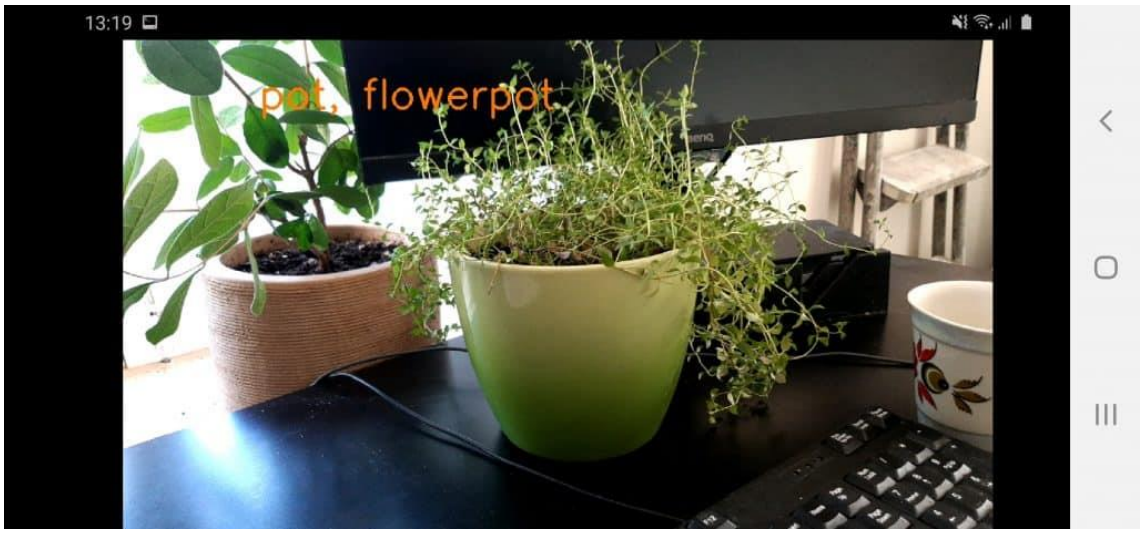
Agora neste tópico será apresentado o trabalho relacionado que aborda a classificação de imagens, destacando sua relevância e contribuições para o campo da visão computacional.

2.4.2.1 Classificação de imagens com *OpenCV* para *Android*

O primeiro a ser abordado será o trabalho realizado por *Murzova* (2020), que apresenta a implementação de um sistema de classificação de imagens utilizando o framework *OpenCV* para o ambiente *Android*. O trabalho descreve como aplicar um modelo de classificação, especificamente com a arquitetura *MobileNetV2*, que de acordo com *Murzova* (2020) essa escolha se baseia na necessidade de realizar a classificação de imagens em tempo real em dispositivos Android, que geralmente possuem recursos de processamento limitados. O trabalho foi desenvolvido no Android Studio utilizando a linguagem Java.

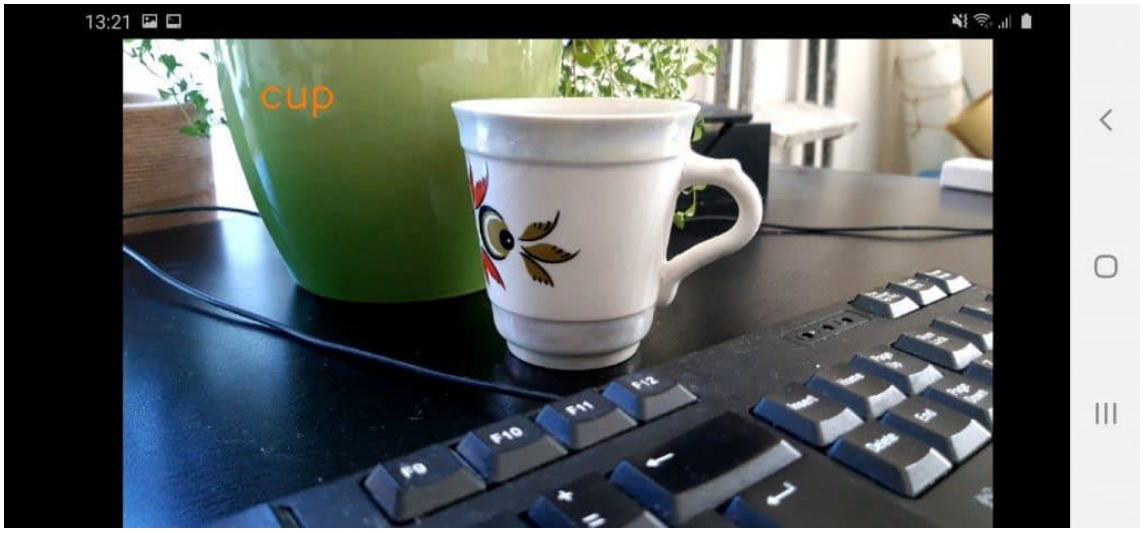
O trabalho de *Murzova* traz contribuições importantes para a classificação de imagens em dispositivos móveis. Sua arquitetura e otimizações foram projetadas para lidar com as restrições de recursos presentes em plataformas móveis, permitindo uma classificação precisa e eficiente mesmo em dispositivos com capacidade de processamento limitada. Essa escolha estratégica possibilita a implementação de aplicativos de classificação de imagens em dispositivos móveis com desempenho satisfatório, como na figura abaixo com o teste ao final do desenvolvimento do sistema.

Figura 5 - Teste *Murzova*



Fonte: *Learn OpenCV - Murzova* (2020).

Figura 6 - Teste Murzova



Fonte: *Learn OpenCV - Murzova* (2020).

Para facilitar a compreensão das distinções e semelhanças entre o presente trabalho e o trabalho anteriormente mencionado, foi elaborada uma tabela de comparação. A Tabela 3 apresenta uma visão geral das características de cada projeto, permitindo uma análise comparativa que auxilia na compreensão das abordagens adotadas, das tecnologias empregadas e das funcionalidades disponibilizadas aos usuários. Essa análise comparativa é de grande importância para obter insights sobre as diferentes estratégias utilizadas, assim como para identificar pontos em comum e possíveis melhorias em ambos os projetos.

Tabela 3 - Diferenças e semelhanças *Murzova*

Características	***EnglishAI***	**Trabalho de *Murzova***
Modelo de Classificação	*MobileNetV2*	*MobileNetV2*
Linguagem de Programação	*Dart (Flutter)*	*Java*
Frameworks Utilizados	*TensorFlow, API Object Detection*	*OpenCV*
Interação com o Ambiente	Captura de imagens	Captura de imagens
Recursos de Usabilidade	Exibição da palavra na imagem e caixa ao redor do objeto	Exibição da palavra na imagem
Ambiente de Desenvolvimento	*Android Studio (Flutter)*	*Android Studio (Java)*

Fonte: Própria autora

Essa tabela comparativa evidencia as diferenças entre os dois trabalhos, ressaltando o uso de diferentes tecnologias, abordagens de usabilidade e recursos disponíveis para o usuário. Ambos os projetos são relevantes no contexto do aprendizado de inglês com base em visão computacional, cada um oferecendo suas próprias características e benefícios distintos.

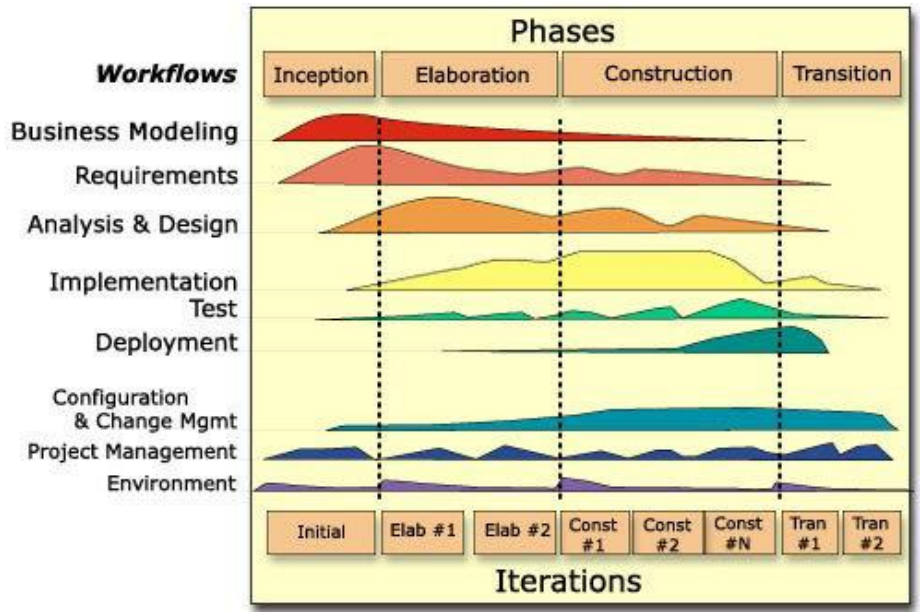
3 METODOLOGIA

Com base no que foi discutido, este capítulo adotará uma abordagem qualitativa, alinhada ao posicionamento de Braga (2004) que destaca a relevância de abordagens qualitativas em pesquisas relacionadas ao desenvolvimento de *software*, especialmente quando se considera a importância da qualidade do sistema. Nesse contexto, para guiar e disciplinar o ciclo de vida do projeto de *software*, será adotado o *Rational Unified Process* (RUP), modelo amplamente reconhecido e abordado por Kruchten (2004). Essa escolha visa estruturar o processo de execução com base no ciclo de desenvolvimento do sistema, conforme as diretrizes e melhores práticas estabelecidas pelo RUP.

3.1 RUP

Com base nas diretrizes estabelecidas pelo RUP, é possível estruturar e disciplinar o ciclo de vida do projeto de software. Ele é um processo iterativo e incremental, que busca atender às demandas da engenharia de software de forma sistemática e controlada (KRUCHTEN, 2004). Ele é composto por fases, cada uma com seus marcos e atividades específicas, que guiam o desenvolvimento do sistema de maneira organizada e consistente. Na Figura 7 é exemplificada suas fases, a primeira dimensão (horizontal) representa o aspecto dinâmico do processo expresso em termos de ciclos, fases, iterações e marcos. No RUP, um produto de software é projetado e construído em uma sucessão de iterações incrementais. Isso permite que testes e validação de ideias de design, assim como mitigação de riscos, ocorram mais cedo no ciclo de vida. A segunda dimensão (vertical) representa o aspecto estático do processo descrito em termos de componentes do processo: atividades, disciplinas, artefatos e papéis (KRUCHTEN, 2004).

Figura 7 - Fases do RUP



Fonte: *Kruchten* (2004).

3.2 Etapas do projeto

Com base no que foi apresentado, visando estruturar o processo de execução com base no ciclo de desenvolvimento do sistema, serão realizadas as seguintes etapas:

- Pesquisa bibliográfica: efetuar pesquisa sobre principais objetos para coleta de imagens; técnicas de reconhecimento das imagens, na sua segmentação e detecção de objetos e trabalhos correlatos; o sistema *android*;
- Definição dos requisitos: Criar e revisar os requisitos, seguindo as necessidades expressadas na pesquisa;
- Definição do modelo de classificação: Pesquisar e definir o modelo de classificação que será implementado e testado, sendo as entradas as imagens dos objetos tirados através do dispositivo móvel;
- Especificação da aplicação: Promover a especificação da aplicação orientada a objetos utilizando a *Unified Modeling Language* (UML). Será utilizada a ferramenta presente no site

diagrams.net para a construção dos diagramas de casos de uso, de classes e de sequência;

- Implementação da aplicação: Para a implementação da aplicação especificada na etapa anterior será utilizado o processo de desenvolvimento de *software* RUP, seguindo a cada incremento no sistema é feita uma validação de risco e um protótipo para refinar a aplicação até um ponto em que esteja pronta, com cada iteração direcionada aos casos de uso. Para desenvolvimento será utilizada a linguagem de programação *Dart* e o *framework flutter*, para o desenvolvimento do modelo de classificação será utilizada a biblioteca de aprendizado de máquina *TensorFlow Lite* e para aplicação o ambiente de desenvolvimento *Android Studio*;
- Testes: Os testes serão por meio de imagens de objetos reais, sendo comparadas com imagens dos objetos presentes no treino do modelo para validar a porcentagem de erro e acerto da ferramenta. Com base nos resultados será feito o refinamento do modelo de classificação.

4 TECNOLOGIAS UTILIZADAS

4.1 *Tensorflow*

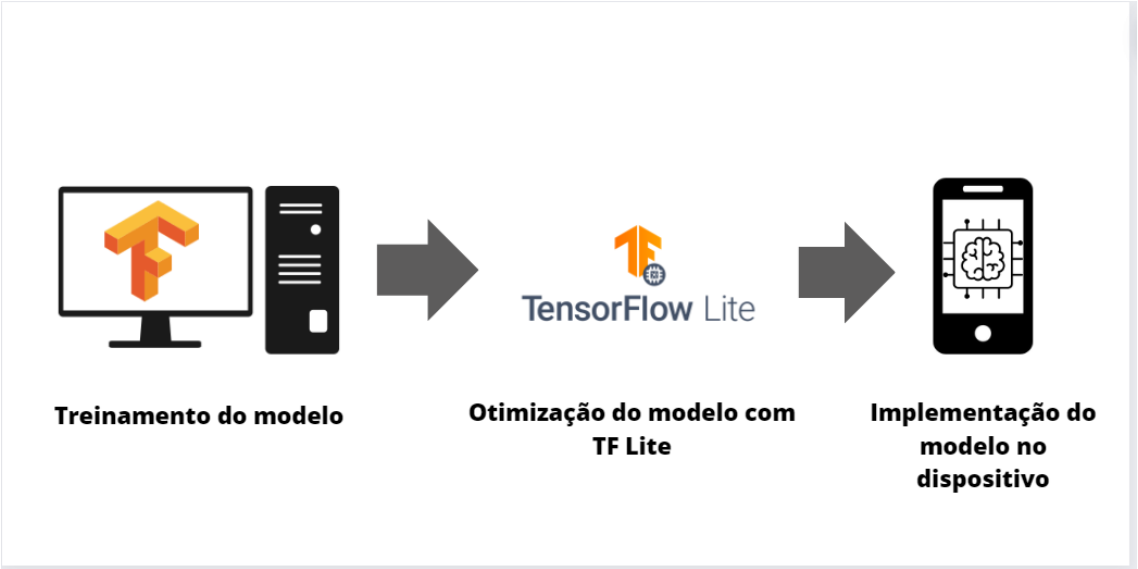
O *TensorFlow* é uma biblioteca de código aberto, desenvolvida pela *Google*, que proporciona um conjunto de ferramentas e recursos para a criação e treinamento de Redes Neurais. Essa biblioteca é altamente versátil, permitindo a aplicação de funções de redes neurais em uma variedade de dispositivos, desde dispositivos móveis até servidores e websites, ele oferece suporte a diversas linguagens de programação, sendo a linguagem *Python* amplamente utilizada para desenvolvimento com essa biblioteca (*TENSORFLOW*, 2015). Além disso, ele pode ser integrado à plataforma de programação *Google Colaboratory*, que é um ambiente de desenvolvimento baseado na nuvem, permitindo a execução e colaboração em projetos de aprendizado de máquina (*ELFATIMI; ERYIGIT; ELFATIMI,* 2022).

4.1.1 *Tensorflow Lite*

O *TensorFlow Lite*, é uma biblioteca específica projetada para implantação em dispositivos móveis e IoT. Essa biblioteca tem como objetivo otimizar modelos desenvolvidos no *TensorFlow*, tornando-os mais leves e rápidos. O *TensorFlow Lite* permite a criação de modelos compactos que podem ser implantados diretamente em aplicativos móveis para realizar inferências em tempo real (*TENSORFLOW LITE*, 2017).

Com ele podemos integrar modelos treinados no *TensorFlow* em aplicativos móveis, permitindo a execução de inferências locais sem a necessidade de conexão com a nuvem. Isso é especialmente útil em cenários em que a latência, a privacidade dos dados ou a disponibilidade de rede são preocupações. A Figura 8 apresenta, de forma resumida, o ciclo de treinamento com *TensorFlow*, ilustrando as etapas do desenvolvimento e implantação de modelos de aprendizado de máquina.

Figura 8 - Ciclo de treinamento



Fonte: Própria autora

4.2 *MobileNetV2*

A arquitetura de rede neural utilizada no modelo de detecção em tempo real foi a *MobileNetV2*, que segundo Sandler (2018), é uma arquitetura de rede neural convolucional projetada especificamente para obter um bom desempenho em dispositivos móveis, levando em consideração as restrições de recursos desses dispositivos.

Desse modo, ao utilizar a *MobileNetV2* como arquitetura de rede neural, foi possível alcançar um equilíbrio entre o desempenho e a eficiência computacional. Isso significa que o modelo resultante é adequado para ser executado em dispositivos móveis, garantindo uma experiência fluída e eficiente para os usuários, além de realizar tarefas de detectção de objetos com alta precisão, combinado com o método de detecção SSD.

4.2.1 *Single Shot Detector*

A abordagem SSD é baseada em uma rede convolucional sequencial que produz uma coleção de caixas delimitadoras de tamanho fixo e pontuações para a presença de instâncias de classes de objetos nessas caixas, seguida por uma etapa de supressão não máxima para gerar as detecções finais. As primeiras camadas da rede são baseadas em uma arquitetura padrão usada para

classificação de imagens de alta qualidade (*MobileNetV2*), depois é adicionado uma estrutura auxiliar (SSD) à rede para produzir detecções (LIU, 2016).

Segundo Liu (2016) cada camada de características adicionada pode produzir um conjunto fixo de previsões de detecção usando um conjunto de filtros convolucionais. Indicados no topo da arquitetura da rede SSD na Figura 9. Para uma camada de características de tamanho m × n com p canais, o elemento básico para prever os parâmetros de uma detecção potencial é um *kernel* pequeno de 3 × 3 × p que produz ou uma pontuação para uma categoria ou um deslocamento de forma em relação às coordenadas da caixa padrão. Em cada uma das m × n localizações onde o kernel é aplicado, ele produz um valor de saída. Os valores de saída de deslocamento da caixa delimitadora são medidos em relação a um padrão predefinido. Então, essas características permitem que a abordagem SSD realize detecções em várias escalas e localize objetos em diferentes partes da imagem.

Figura 9 - SSD Mobilenet

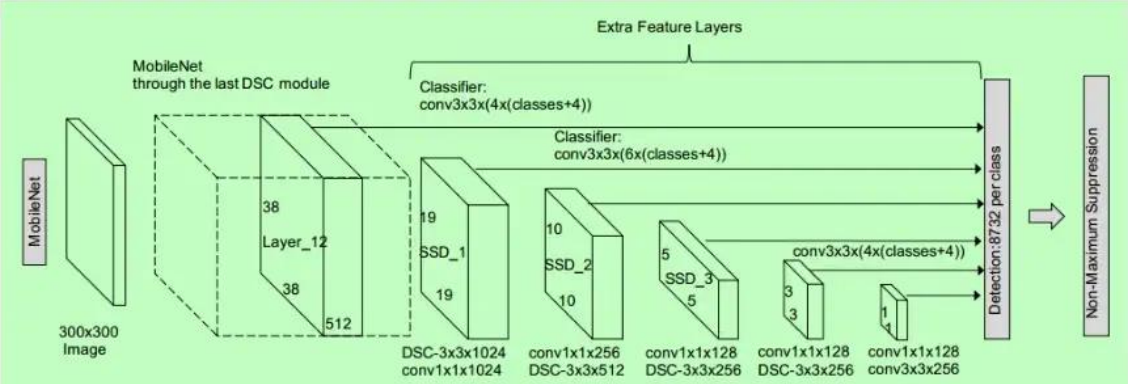


Figure 6. Mobile-Det: SSD-based detection with MobileNet as backbone, modified based on [14]

Fonte: *Singhal* (2020)

Com esse método, é dividido o *kernel* em dois *kernels* de formas 3 x 1 e 1 x 3. A entrada é primeiro convolucionada com 3 x 1 *kernel* e então com 1 x 3, assim o número de parâmetros seria 3 + 3 = 6, ao invés dos 9 inicialmente. Portanto, uma menor operação de multiplicação de matrizes é necessária, melhorando o processamento. Em conclusão, a implementação bem-sucedida do SSD *MobilenetV2* nos permite obter detecção em tempo real de objetos utilizando visão computacional. No entanto, para alcançar resultados precisos e abrangentes, é essencial contar com conjuntos de dados robustos e

diversificados. O próximo tópico abordará o COCO *dataset*, uma referência amplamente reconhecida nesse campo.

4.3 *COCO Dataset*

O COCO é um conjunto de dados abrangente utilizado para tarefas de detecção de objetos, segmentação e geração de legendas. Criado com o objetivo de avançar a pesquisa em visão computacional, ele contém milhares de imagens rotuladas com anotações detalhadas sobre objetos presentes nas cenas. (COCO, 2021).

Ele utiliza um formato JSON que fornece informações sobre cada conjunto de dados e todas as imagens nele, incluindo licenças de imagem, categorias usadas para classificar objetos nas imagens, dados brutos da imagem como tamanho de pixel e as anotações importantes dos objetos presentes nas imagens através da segmentação de instância para anotar os objetos nas imagens. Esse tipo de segmentação atribui uma máscara única a cada objeto individualmente, permitindo uma identificação precisa e separada de diferentes instâncias de objetos na imagem. (DATAGEN, 2011).

Portanto, a escolha de utilizar o COCO *dataset* é para garantir a eficácia e a generalização do nosso sistema de detecção de objetos. Ao utilizar um conjunto de dados tão abrangente e amplamente utilizado, estamos capacitando nosso aplicativo a reconhecer e compreender uma ampla variedade de objetos, proporcionando uma experiência mais rica e precisa para os usuários.

4.4 *Open Image Dataset*

O *Open Images Dataset* é um conjunto de dados aberto e de larga escala que foi criado para fins de pesquisa e desenvolvimento em visão computacional. Ele contém milhões de imagens anotadas em diversas categorias, abrangendo uma ampla variedade de objetos e cenários do mundo real. O objetivo dele é fornecer um recurso valioso para treinamento e avaliação de algoritmos de visão computacional. Essas imagens e suas anotações podem ser utilizadas para tarefas como detecção de objetos, classificação de imagens e segmentação semântica. (KUZNETSOVA, 2018).

Este *dataset* foi escolhido como o conjunto de dados para o modelo de classificação devido à sua vasta variedade de classes e ao grande número de imagens disponíveis para cada uma delas. Essa seleção visa aprimorar a eficiência e a qualidade dos dados, uma vez que as imagens abrangem uma ampla gama de cenários e contextos. Essa diversidade de cenários permite que o modelo seja treinado em diferentes condições e seja capaz de reconhecer objetos em diversos ambientes.

4.5 *Teachable Machine*

Com base nisso, para aplicar este *dataset* no modelo de classificação foi utilizada a ferramenta *Teachable Machine* que é feito pela *Google e* que permite a criação rápida de modelos de aprendizado de máquina sem programação. É uma maneira de aprender os conceitos fundamentais do aprendizado de máquina e ser capaz de aplicá-los a projetos sem ter que entender a matemática complexa normalmente necessária para trabalhar com o aprendizado de máquina. (STANCIU, 2022).

A ferramenta é na *web* e possui aplicação para projetos com imagens, sons e poses, além de serem feitos em *Tensorflow,* podendo ser exportados e implementados em aplicativos móveis, sistemas embarcados e em páginas *web.* (TEACHABLE MACHINE, 2018).

4.6 *Flutter*

Flutter é um framework de código aberto desenvolvido pela *Google* para criar aplicativos bonitos, compilados nativamente e multiplataforma a partir de um único código-fonte (*FLUTTER*, 2015).

Uma das principais vantagens do *Flutter* é a sua ampla variedade de widgets personalizados, que proporcionam uma interface leve e uma experiência de usuário excelente. De acordo com a *Google*, o *Flutter* é capaz de exibir animações suaves a 60 quadros por segundo e oferece um desempenho quase tão bom quanto um aplicativo nativo (*CAPPELI*, 2018).

4.7 *Dart*

Dart é uma linguagem de programação orientada a objetos otimizada para o desenvolvimento rápido de aplicativos. Seu objetivo é fornecer maior produtividade no desenvolvimento multiplataforma, em conjunto com um tempo de execução flexível para frameworks de aplicativos (*DART*, 2011).

5 PROJETO DO SISTEMA

5.1 Introdução do Sistema

O presente capítulo tem como objetivo detalhar o desenvolvimento do sistema, com foco na criação de uma interface adequada para crianças e integração dos modelos. A tela inicial da aplicação apresenta opções de aprendizado, como a possibilidade de enviar uma nova imagem para obter a pronúncia e a palavra em inglês correspondente ou realizar a detecção do conteúdo da imagem em tempo real.

5.2 Análise e Especificação dos Requisitos

Conforme os resultados da pesquisa prévia, verificou-se que as ferramentas mais adequadas para implementação em um aplicativo móvel seriam o *Tensorflow* (API *Object detection e Teachable Machine*), desse modo para implementação da aplicação foram estabelecidos os seguintes requisitos:

- O usuário deve poder abrir a câmera do seu dispositivo móvel e obter o resultado de detecção de objetos (Requisito Funcional – RF);
- O usuário deve poder tirar uma foto de um objeto a partir da câmera do seu dispositivo móvel (Requisito Funcional – RF);
- O sistema deve armazenar a foto para processamento (Requisito Funcional – RF);
- O modelo deve segmentar a imagem tirada pelo usuário, a fim de extrair o ponto de interesse que é o objeto a ser traduzido (Requisito Funcional - RF);
- O modelo deve classificar a imagem de acordo com as imagens de treino (Requisito Funcional - RF);
- A aplicação deve informar o resultado da classificação (palavra em inglês) de forma visual e auditiva;
- A aplicação deve conter uma interface agradável e intuitiva para o público-alvo (Requisito Funcional - RF);

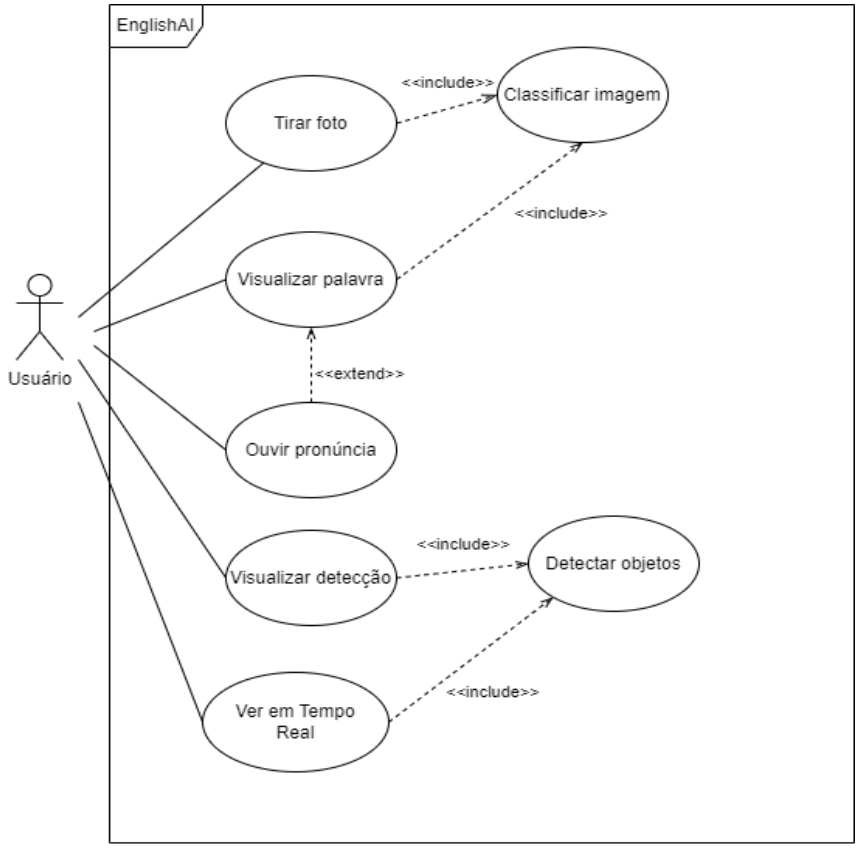
- O resultado da classificação deve ser mostrado em até no máximo 20 segundos (Requisito Não Funcional - RNF);

5.3 Casos de Uso

5.3.1 Diagrama de caso de uso

A Figura 10 apresenta o diagrama de caso de uso do sistema, sendo o ator principal o Usuário, que interage com o sistema em duas formas distintas: Tirar Foto e Imagem em Tempo Real. No caso de Tirar Foto, o usuário captura uma imagem, que é classificada pelo modelo. Em seguida, a classificação da imagem é exibida e o usuário tem a opção de ouvir a pronúncia da palavra correspondente. Já na opção de Imagem em Tempo Real, a câmera é aberta e o sistema realiza a detecção de objetos em tempo real, exibindo caixas delimitadoras ao redor dos objetos detectados.

Figura 10 - Diagrama de Casos de Uso



Fonte: Própria autora

5.3.2 Descrição dos Casos de Uso

Tabela 4 - Descrição do caso de uso “Tirar Foto”

Caso de uso	UC01 – Tirar Foto
Descrição	O usuário poderá abrir a câmera e enviar uma foto para o sistema
Atores	Usuário
Pré-condição	• O usuário deve permitir o uso da câmera.
Pós-condições	• O sistema deve apresentar a foto capturada pelo usuário em pelo menos 5 segundos
Fluxo principal	1. Escolher opção de tirar foto 2. Tela de permissão de acesso a câmera 3. Aceitar 4. Clicar no botão de tirar foto 5. Acessar tela com a foto capturada.
Fluxo alternativo	1. Escolher opção de tirar foto 2. Tela de permissão de acesso a câmera 3. Negar 4. Entra em looping até aceitar permissão

Fonte: Própria autora

Tabela 5 - Descrição do caso de uso “Classificar imagem”

Caso de uso	UC02 – Classificar imagem
Descrição	O sistema utiliza o modelo para classificar a imagem
Atores	Sistema
Pré-condição	• O modelo deve estar no sistema • O usuário deverá ter tirado uma foto

Pós-condições	• Atualiza a variável com a resposta do modelo
Fluxo principal	1. O sistema recebe a foto capturada 2. O sistema utiliza o modelo para classificar a imagem 3. O sistema atualiza a variável com a resposta do modelo e seta o estado da tela
Fluxo alternativo	-

Fonte: Própria autora

Tabela 6 - Descrição do caso de uso “Visualizar palavra”

Caso de uso	UC03 – Visualizar palavra
Descrição	O usuário poderá visualizar a classificação retornada pelo modelo
Atores	Usuário, Sistema
Pré-condição	• O modelo deverá ter retornado à classificação da imagem
Pós-condições	• A classificação é exibida na tela
Fluxo principal	1. O sistema recebe a classificação 2. O sistema exibe a classificação e a imagem para o usuário 3. O usuário visualiza a palavra em inglês
Fluxo alternativo	-

Fonte: Própria autora

Tabela 7 - Descrição do caso de uso “Ouvir pronúncia”

Caso de uso	UC04 – Ouvir pronúncia

Descrição	O usuário tem a opção de ouvir a pronúncia da palavra classificada
Atores	Usuário
Pré-condição	• Classificação da imagem exibida
Pós-condições	• Pronúncia da palavra ouvida
Fluxo principal	1. O usuário seleciona a opção "Ouvir Pronúncia" 2. O sistema reproduz a pronúncia em inglês correspondente à palavra
Fluxo alternativo	-

Fonte: Própria autora

Tabela 8 - Descrição do caso de uso “Ver em Tempo Real”

Caso de uso	UC05 – Ver em Tempo Real
Descrição	O usuário poderá visualizar a detecção dos objetos com a câmera aberta
Atores	Usuário
Pré-condição	• O usuário deve permitir o uso da câmera.
Pós-condições	• Objetos detectados exibidos em tempo real
Fluxo principal	1. O usuário seleciona a opção "Imagem em Tempo Real" 2. O sistema abre a câmera do dispositivo 3. O sistema carrega o modelo SSD MobileNetV2 COCO previamente treinado.

Fluxo alternativo	1. Escolher opção de "Imagem em Tempo Real" 2. Tela de permissão de acesso a câmera 3. Negar 4. Entra em looping até aceitar permissão

Fonte: Própria autora

Tabela 9 - Descrição do caso de uso “Detectar objetos”

Caso de uso	UC06 – Detectar objetos
Descrição	O sistema utiliza o modelo para detectar os elementos que passam pela câmera
Atores	Sistema
Pré-condição	• Câmera do dispositivo em funcionamento • Modelo deve estar no sistema
Pós-condições	• Objetos detectados exibidos
Fluxo principal	1. O sistema inicia a detecção de objetos em tempo real, aplicando o modelo às imagens capturadas pela câmera 2. O sistema analisa cada quadro da imagem em tempo real e identifica a presença de objetos dentro dele. 3. Caso um objeto seja detectado, o sistema desenha uma caixa delimitadora ao redor do objeto e a palavra em inglês 4. O sistema continua a atualizar a exibição da detecção à medida que novos quadros são capturados pela câmera.
Fluxo alternativo	-

Fonte: Própria autora

Tabela 10 - Descrição do caso de uso “Visualizar detecção”

Caso de uso	UC07 – Visualizar detecção
Descrição	O usuário visualiza a detecção de objetos exibida pelo sistema
Atores	Usuário

Pré-condição	• Modelo já funcionando
Pós-condições	• Visualização da detecção
Fluxo principal	1. Passar a câmera do dispositivo ao redor 2. Visualizar a exibição do resultado da detecção em tempo real na tela do dispositivo, destacando os objetos encontrados.
Fluxo alternativo	-

Fonte: Própria autora

5.4 Diagrama de Classes

O diagrama de classes a seguir apresenta as classes presentes no sistema, incluindo seus atributos, métodos e as relações entre elas. Ele visa proporcionar uma visão estruturada das principais entidades do sistema e como elas interagem umas com as outras.

A Figura 11, mostra como as classes do sistema interagem, sendo a classe *main* a classe principal do aplicativo, responsável por iniciar a execução da aplicação *Flutter*. Essa classe é responsável por garantir a inicialização correta dos widgets *(WidgetsFlutterBinding.ensureInitialized())* e por executar a função *runApp*() para iniciar o aplicativo, essa função chama a classe *MyApp* que é um *StatelessWidget* que representa o aplicativo como um todo. Ela define o tema do aplicativo, o título e a tela inicial (*home*). O método *build (BuildContext)* é responsável por construir a interface do aplicativo, no *home* é chamada a classe *HomePage* que é uma *StatefulWidget* responsável por exibir a tela inicial do aplicativo. Ela contém uma lista de descrições de câmeras (*cameras*) e métodos para carregar as câmeras disponíveis (*_loadCameras*()) e construir a interface (*build(BuildContext)*). A classe também possui uma estrutura de *layout* com imagens e botões, nessa tela tem dois botões que podem chamar classes distintas, sendo o primeiro chama a classe *ObjectDetection* que é um *StatefulWidget* que representa a tela de detecção de objetos. Ela possui uma propriedade *cameras* que armazena a lista de câmeras disponíveis. O método *createState*() retorna uma instância da classe *_ObjectDetectionState* que é um *State* de *ObjectDetection* e é responsável por gerenciar o estado da tela de detecção de objetos. Ela possui as variáveis *_recognitions*, *_imageHeight*, *_imageWidth* e *_model*, que armazenam informações sobre as detecções realizadas. Os métodos *initState*(), *loadModel*() e *setRecognitions*() são responsáveis por carregar o modelo de detecção, definir as detecções reconhecidas e atualizar a interface do usuário. O método *build(BuildContext)* constrói a interface da tela, utilizando o *widget Camera* que é um *StatefulWidget* que representa a visualização da câmera no aplicativo. Ela possui as propriedades *cameras*, que armazena a lista de câmeras disponíveis, e *setRecognitions*, que é uma função de retorno de chamada para definir as detecções reconhecidas. O método *createState*() retorna uma instância da

classe *_CameraState*, ele é responsável por gerenciar o estado da visualização da câmera. Possui as variáveis *controller*, que é responsável por controlar a câmera, e *isDetecting*, que indica se o sistema está realizando a detecção de objetos no momento. O método *initState*() é chamado quando o estado é inicializado e é responsável por inicializar a câmera e configurar a detecção de objetos. O método *dispose*() é chamado quando o estado é descartado e é responsável por liberar os recursos da câmera. O método *_detectObjectOnFrame(CameraImage)* é responsável por executar a detecção de objetos no quadro da câmera e o método *build(BuildContext)* é responsável por construir a interface da visualização da câmera.

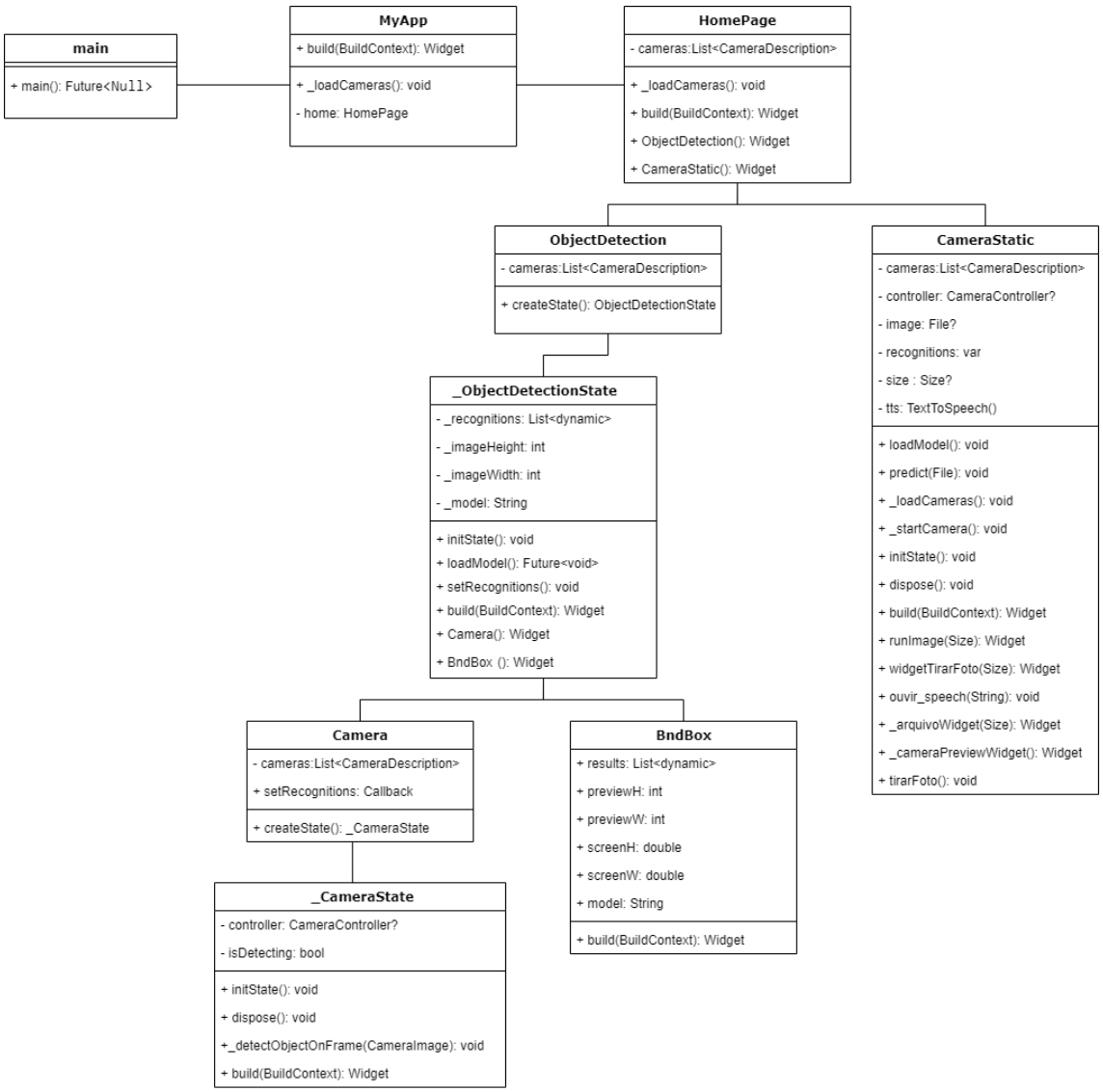
Já o *widget BndBox* é um *StatelessWidget* responsável por renderizar as caixas delimitadoras dos objetos detectados. Ela possui as propriedades *results*, que armazena a lista de resultados da detecção de objetos, *previewH* e *previewW*, que representam a altura e a largura da visualização da câmera, *screenH* e *screenW*, que representam a altura e a largura da tela do dispositivo, e *model*, que representa o modelo utilizado para a detecção. O método *build(BuildContext)* é responsável por construir a interface das caixas delimitadoras com base nos resultados da detecção.

Agora quando o usuário opta por tirar foto é chamada a classe *CameraStatic* que também é um *StatefulWidget* e é responsável por lidar com a captura de imagens estáticas e a classificação delas. Ela possui propriedades como *cameras*, que armazena a lista de descrições das câmeras disponíveis, *controller*, que controla a câmera, *image*, que armazena a imagem capturada, *recognitions*, que armazena os resultados da classificação, *size*, que representa o tamanho da tela do dispositivo, e *tts*, que é uma instância da classe *TextToSpeech* para reprodução de áudio.

Possui também, métodos como *loadModel*, responsável por carregar o modelo de classificação, *predict*, responsável por realizar a classificação na imagem, *_loadCameras*, responsável por carregar as câmeras disponíveis, *_startCamera*, responsável por iniciar a câmera, *initState* e *dispose*, responsáveis por inicializar e descartar a classe, respectivamente. Assim, o método *build* constrói a interface da câmera, exibindo a imagem capturada ou a visualização da câmera, dependendo da variável *flag*. Os métodos *runImage*,

widgetTirarFoto, *widgetPredicted, ouvir_speech, _arquivoWidget, _cameraPreviewWidget* e tirarFoto são responsáveis por construir os elementos da interface e realizar as ações correspondentes.

Figura 11 - Diagrama de Classes



Fonte: Própria autora

6 SISTEMA

O presente capítulo apresentado neste documento visa explicar como o sistema foi implementado, a implementação foi dividida em duas etapas, a primeira delas foi a criação dos dois modelos utilizados na aplicação, sendo o primeiro um modelo de detecção de imagens em tempo real e o segundo um modelo de classificação de imagens. A segunda etapa é o processo de desenvolvimento da própria aplicação mobile, a qual está com os modelos já embarcados e seguindo uma interface conforme a metodologia pesquisada.

6.1 Detecção em tempo real

Nesta etapa, foi desenvolvido o código para a criação do modelo de detecção de objetos utilizando a API de detecção de objetos do *TensorFlow*, especificamente o modelo *ssd_mobilenetv2_coco*. De acordo com *Singhal* (2020), esse modelo é um *framework* eficaz para criar uma rede de aprendizado profundo capaz de resolver problemas de detecção de objetos, permitindo detectar múltiplos objetos em um único quadro com precisão e eficiência. Desse modo, a utilização dele demonstra-se vantajosa devido à sua capacidade de detectar e identificar diversos objetos em tempo real, tornando-o uma escolha valiosa para a tarefa de detecção de objetos em diferentes cenários.

Para isso começo instalando o *object detection* com o comando apresentado na Figura 12.

Figura 12 - Instalação *Object Detection*

```
!pip install object_detection
```

Fonte: Própria autora

Depois que o diretório é carregado no ambiente é criada a função para acessar o modelo, na figura 13, em que ela recebe o nome do modelo como parâmetro e realiza os seguintes passos:

- Define o URL base para o download do modelo a partir do diretório do *TensorFlow* carregado.

- Concatena o nome do arquivo do modelo com a URL base para obter o arquivo completo para *download*.
- Utiliza a função "*get_file*" da biblioteca *tf.keras.utils* para fazer o *download* do modelo especificado a partir do URL.
- Descompacta o arquivo baixado para obter o diretório do modelo.
- Carrega o modelo descompactado utilizando a função *"tf.saved_model.load"*, que retorna o modelo carregado.
- Obtém a assinatura *'serving_default'* do modelo, que é a assinatura padrão utilizada para inferência em modelos salvos pelo *TensorFlow Serving.*

Após executar essas etapas, a função retorna o modelo pronto para ser utilizado.

Figura 13 - Função de carregamento do modelo

```
def load_model(model_name):
  base_url = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/'
  model_file = model_name + '.tar.gz'
  model_dir = tf.keras.utils.get_file(
    fname=model_name,
    origin=base_url + model_file,
    untar=True)

  model_dir = pathlib.Path(model_dir)/"saved_model"

  model = tf.saved_model.load(str(model_dir))
  model = model.signatures['serving_default']

  return model
```

Fonte: Própria autora

Após a criação da função "*load_model*", o código continua com a definição de algumas variáveis adicionais necessárias, mostradas na Figura 14, são elas:

- *PATH_TO_LABELS*: Essa variável armazena o caminho para o arquivo *"mscoco_label_map.pbtxt",* que contém o mapeamento das classes de objetos que o modelo é capaz de detectar. Esse arquivo especifica a relação entre os *IDs* numéricos das classes e seus respectivos nomes. O arquivo está localizado na pasta *"object_detection/data/"*.

- *category_index:* Esta variável utiliza a função "*create_category_index_from_labelmap*" do módulo "*label_map_util"* da API de detecção de objetos do *TensorFlow*. Essa função lê o arquivo "*mscoco_label_map.pbtxt*" e cria um índice que mapeia os *IDs* numéricos das classes para seus nomes. O parâmetro "*use_display_name=True*" indica que os nomes de exibição das classes serão utilizados quando disponíveis.

Com essas variáveis definidas, o código está pronto para realizar a detecção de objetos usando o modelo carregado e o índice das classes. Isso permite identificar os objetos presentes nas imagens e atribuir seus respectivos rótulos com base nas classes definidas no arquivo *"mscoco_label_map.pbtxt"*. Na Figura 15 há o que a variável *category_index* criou, mostrando as primeiras 20 linhas.

Figura 14 - Variáveis adicionais

```
PATH_TO_LABELS = 'object_detection/data/mscoco_label_map.pbtxt'

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

Fonte: Própria autora

Figura 15 - *category_index*

```
{1: {'id': 1, 'name': 'person'},
 2: {'id': 2, 'name': 'bicycle'},
 3: {'id': 3, 'name': 'car'},
 4: {'id': 4, 'name': 'motorcycle'},
 5: {'id': 5, 'name': 'airplane'},
 6: {'id': 6, 'name': 'bus'},
 7: {'id': 7, 'name': 'train'},
 8: {'id': 8, 'name': 'truck'},
 9: {'id': 9, 'name': 'boat'},
 10: {'id': 10, 'name': 'traffic light'},
 11: {'id': 11, 'name': 'fire hydrant'},
 13: {'id': 13, 'name': 'stop sign'},
 14: {'id': 14, 'name': 'parking meter'},
 15: {'id': 15, 'name': 'bench'},
 16: {'id': 16, 'name': 'bird'},
 17: {'id': 17, 'name': 'cat'},
 18: {'id': 18, 'name': 'dog'},
 19: {'id': 19, 'name': 'horse'},
 20: {'id': 20, 'name': 'sheep'},
```

Fonte: Própria autora

6.1.1 Teste do modelo

Como o foco deste trabalho é utilizar um modelo de visão computacional pré-treinado para aprimorar o aprendizado de inglês. Para verificar a eficácia do modelo, foi realizado um único teste inicial para avaliar sua funcionalidade e precisão, com isso foi criado um código para teste, iniciando na Figura 16, com o carregamento das imagens de teste, a variável *PATH_TO_TEST_IMAGES_DIR* representa o diretório no qual as imagens de teste estão armazenadas.

Posteriormente, a variável *TEST_IMAGE_PATHS* é criada para armazenar os caminhos das imagens de teste, utilizando o método *glob*() da biblioteca *pathlib*. Esse método é responsável por listar todas as imagens com extensão *.jpg* presentes no diretório de teste. Os caminhos são armazenados em uma lista e, em seguida, são ordenados alfabeticamente para garantir uma sequência consistente durante os testes. Com a lista de caminhos das imagens de teste pronta, é possível iterar sobre ela para realizar a detecção de objetos nas imagens usando o modelo de visão computacional selecionado, como mencionado anteriormente. Esse processo permite avaliar o desempenho do modelo em diferentes contextos visuais e aferir a sua precisão em detectar objetos específicos presentes nas imagens de teste.

Figura 16 - Carregamento imagens de teste

```
PATH_TO_TEST_IMAGES_DIR = pathlib.Path('/content/drive/MyDrive/DT&ML/TCC/Banco_de_imagens/Teste')
TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))
```

Fonte: Própria autora

Após obter as imagens de teste e carregar o modelo pré-treinado, são criadas duas funções importantes para executar a detecção de objetos nas imagens e exibir os resultados, mostradas na Figura 17 e 18, são elas:

- A função *run_inference_for_single_image(model, image)* realiza a inferência de detecção de objetos em uma única imagem usando o modelo carregado. Essa função recebe o modelo e uma imagem como entrada e retorna um dicionário contendo os resultados da detecção. O processo consiste em converter a imagem para um tensor, passá-la através do modelo, processar as saídas do modelo para obter as

detecções, e refinar as máscaras de detecção, se estiverem presentes, os resultados são então organizados em um dicionário e retornados pela função.

- A função *show_inference(model, image_path, class_id)* é responsável por exibir visualmente os resultados da detecção de objetos em uma imagem específica. Essa função recebe o modelo, o caminho da imagem a ser testada e um identificador de classe como entrada. Primeiro, a imagem é convertida em um *array numpy*. Em seguida, a função *run_inference_for_single_image()* é chamada para realizar a detecção de objetos na imagem. Os resultados são filtrados para manter apenas as detecções com uma confiança (score) acima de 0.5. As caixas delimitadoras, as classes detectadas e os scores são extraídos dos resultados e, finalmente, a função *visualize_boxes_and_labels_on_image_array()* é chamada para desenhar as caixas delimitadoras e rótulos na imagem original, destacando os objetos detectados.

Essas funções são essenciais para a análise dos resultados do modelo de detecção de objetos em tempo real ou nas imagens de teste, permitindo a visualização das detecções realizadas e a avaliação da precisão e desempenho do modelo.

Figura 17 - Funções para teste

```
def run_inference_for_single_image(model, image):
  image = np.asarray(image)
  input_tensor = tf.convert_to_tensor(image)
  input_tensor = input_tensor[tf.newaxis,...]
  output_dict = model(input_tensor)
  num_detections = int(output_dict.pop('num_detections'))
  output_dict = {key:value[0, :num_detections].numpy()
                 for key,value in output_dict.items()}
  output_dict['num_detections'] = num_detections
  output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

  if 'detection_masks' in output_dict:
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
              output_dict['detection_masks'], output_dict['detection_boxes'],
               image.shape[0], image.shape[1])
    detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                       tf.uint8)
    output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

  return output_dict
```

Fonte: Própria autora

Figura 18 - Funções para teste

```
def show_inference(model, image_path,class_id):
  image_np = np.array(Image.open(image_path))
  output_dict = run_inference_for_single_image(model, image_np)
  boxes = []
  classes = []
  scores = []
  for i,x in enumerate(output_dict['detection_classes']):
    if output_dict['detection_scores'][i] > 0.5:
      classes.append(x)
      boxes.append(output_dict['detection_boxes'][i])
      scores.append(output_dict['detection_scores'][i])
  boxes = np.array(boxes)
  classes = np.array(classes)
  scores = np.array(scores)
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      boxes,
      classes,
      scores,
      category_index,
      instance_masks=output_dict.get('detection_masks_reframed', None),
      use_normalized_coordinates=True,
      line_thickness=4)

  display(Image.fromarray(image_np))
```

Fonte: Própria autora

Após isso é iniciado o teste, na Figura 19 é mostrado a atribuição do modelo a uma variável e inicia um *loop* que percorre todas as imagens presentes no diretório de teste (*TEST_IMAGE_PATHS).*

Dentro do *loop*, a função *show_inference()* é chamada para cada imagem, passando o modelo de detecção, o caminho da imagem e um identificador de classe como parâmetros, neste caso o identificador não filtra, ele traz todas as classes que o modelo identificar, mesmo se colocar 17, que corresponde a classe de gato.

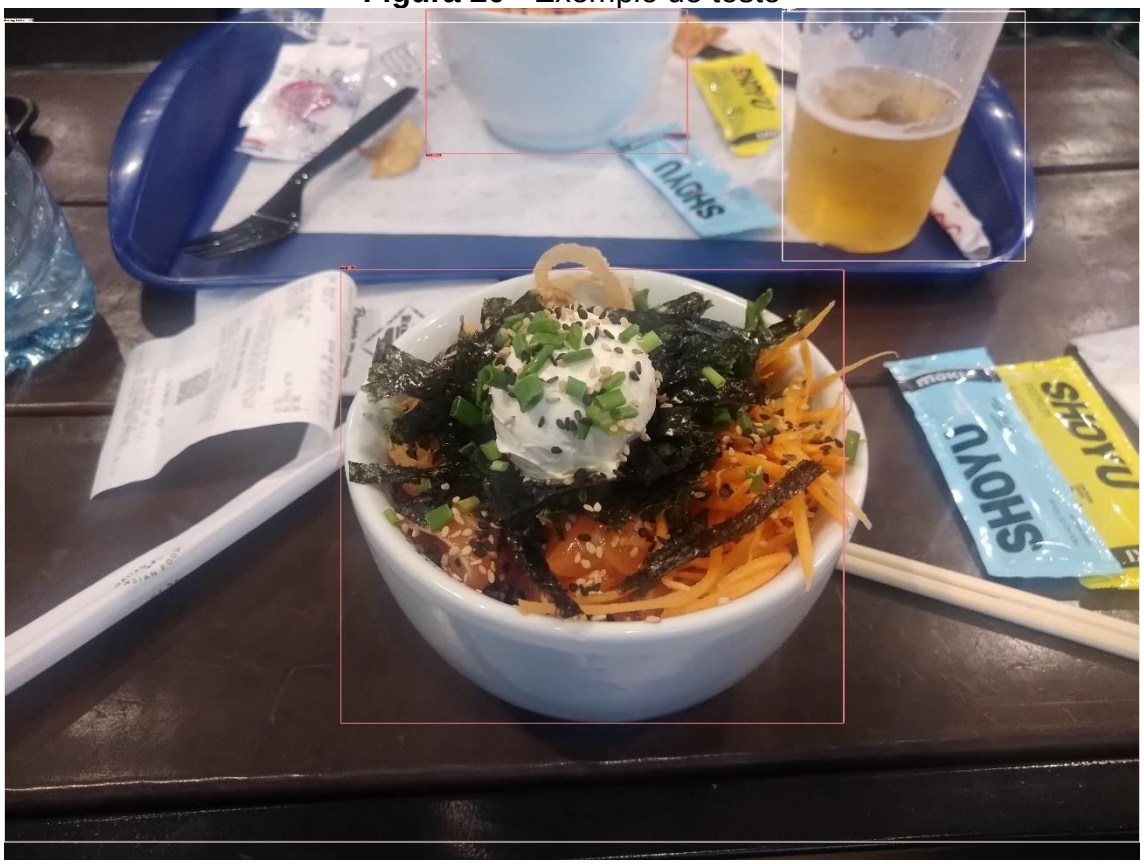
Dito isso, a função *show_inference()* realiza a detecção de objetos na imagem e visualiza os resultados, desenhando as caixas delimitadoras e rótulos na imagem, destacando os objetos detectados, como mostrado na Figura 20.

Figura 19 - Início do teste

```
detection_model = load_model(model_name)
for image_path in TEST_IMAGE_PATHS:
  show_inference(detection_model, image_path, 17)
```

Fonte: Própria autora

Figura 20 - Exemplo de teste



Fonte: Própria autora

6.1.2 Resultado do teste

Durante o teste, verificou-se que o modelo era capaz de detectar corretamente uma variedade de objetos em tempo real. Na Figura 21 contém os resultados do teste, sendo utilizadas 15 imagens, cada imagem continha vários objetos que o modelo tinha a capacidade de detectar, totalizando 72 objetos distintos (classes), sendo possível que algumas classes se repetissem em diferentes imagens (por exemplo, duas pessoas em uma mesma imagem).

Durante o teste, o modelo foi capaz de detectar corretamente 56% dos objetos presentes nas imagens. Dos objetos detectados, 95% foram classificados corretamente, enquanto 5% foram classificados erroneamente. Esse desempenho resultou em uma taxa geral de precisão do modelo de 53%.

Figura 21 - Resultado do teste

Imagens	Objetos	Detectou	Acertou	Errou	Precisão
15	72	56%	95%	5%	53%

Fonte: Própria autora

No entanto, é válido ressaltar que a precisão geral de 53% indica que o modelo pode ser aprimorado em pesquisas futuras, ou seja, esse teste demonstra o potencial do modelo em detectar objetos em um cenário real, mas também revela a importância contínua da pesquisa e desenvolvimento para aprimorar as capacidades de detecção de objetos, visando a aplicação em diversos contextos práticos e aplicações específicas.

6.1.3 Conversão do modelo

Para viabilizar a utilização do modelo de detecção de objetos em um aplicativo móvel, foi necessário realizar a conversão do modelo original para um formato otimizado, adequado ao ambiente dos dispositivos móveis. Esse processo de conversão foi realizado por meio do *TensorFlow Lite*, como citado anteriormente é uma versão otimizada do *TensorFlow*, especialmente projetada para aplicações em dispositivos móveis e sistemas embarcados.

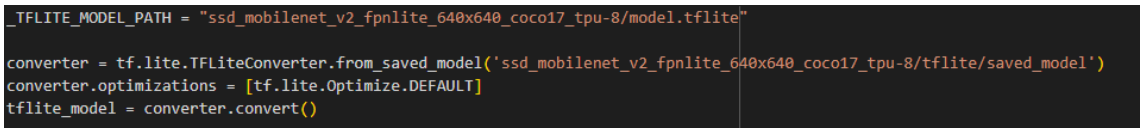
Na Figura 22, o código inicia especificando o caminho para o modelo salvo que foi previamente treinado para a detecção de objetos. Em seguida, é criado um objeto *TFLiteConverter*, responsável por realizar a conversão do modelo. Utilizando a função *from_saved_model*, o modelo é carregado em memória.

Para otimizar ainda mais o modelo convertido, é aplicada a otimização *Optimize.DEFAULT*, que aplica uma configuração padrão de otimizações para melhorar a eficiência e o desempenho do modelo no formato *.tflite*. Essas otimizações garantem que o modelo ocupe menos espaço em memória e seja executado de forma mais rápida e eficiente nos dispositivos móveis.

Por fim, a conversão é realizada utilizando o método *convert*() do objeto *TFLiteConverter*, que retorna o modelo convertido no formato *.tflite*. O modelo otimizado é armazenado na variável *tflite_model*, pronto para ser integrado ao aplicativo móvel.

Essa etapa de conversão é de extrema importância, pois permite que o modelo seja facilmente incorporado ao aplicativo móvel, proporcionando a funcionalidade de detecção de objetos em tempo real. Através do formato *.tflite*, o modelo se torna mais leve e eficiente, garantindo uma experiência mais ágil e responsiva para o usuário durante o uso do aplicativo. Essa abordagem demonstra a relevância de adaptar modelos complexos para formatos otimizados, a fim de viabilizar sua integração em aplicativos móveis e melhorar a experiência do usuário com a detecção precisa e eficiente de objetos em tempo real.

Figura 22 - Conversão do modelo



Fonte: Própria autora

6.2 Tirando a foto do objeto – Modelo de classificação

Nesta seção, descreve-se a implementação do modelo de classificação usando a ferramenta *Teachable Machine*. Esse modelo permite que os usuários tirem uma foto de um objeto e obtenham a palavra correspondente em inglês, além de poderem ouvir a pronúncia. A escolha das classes do modelo foi baseada em uma pesquisa prévia, levando em consideração os elementos do ambiente em que a criança vive.

6.2.1 Obtenção e tratamento das imagens

Nesta etapa foi feita a obtenção das imagens, elas foram provenientes do *Open Image dataset v4,* por conter uma variedade de classes, foi buscando nele classes que estivessem no contexto de vida da criança. Essa abordagem permitiu um modelo mais direcionado, fornecendo resultados mais significativos e úteis para a aprendizagem interativa de inglês.

Para coletar as imagens foi feito um código para baixar as imagens que utilizaria deste *dataset,* seguindo os passos de *Nayak* (2019), para poder selecionar apenas as classes válidas para este trabalho e mudar alguns parâmetros. Há formas de adicionar parâmetros opcionais para excluir determinados tipos de imagens, definindo-os explicitamente como 0 na linha de comando. (*NAYAK* 2019).

Na Figura 23, podemos observar o processo de *download* da classe "*Window*" do conjunto de treinamento, porém, com a exclusão de imagens que apresentam múltiplas classes em uma única imagem (*groupOf*) e imagens com perspectiva interna, como no caso de um carro fotografado do seu interior (*inside*). Essas exclusões foram feitas para garantir que as imagens coletadas fossem mais adequadas ao propósito do trabalho.

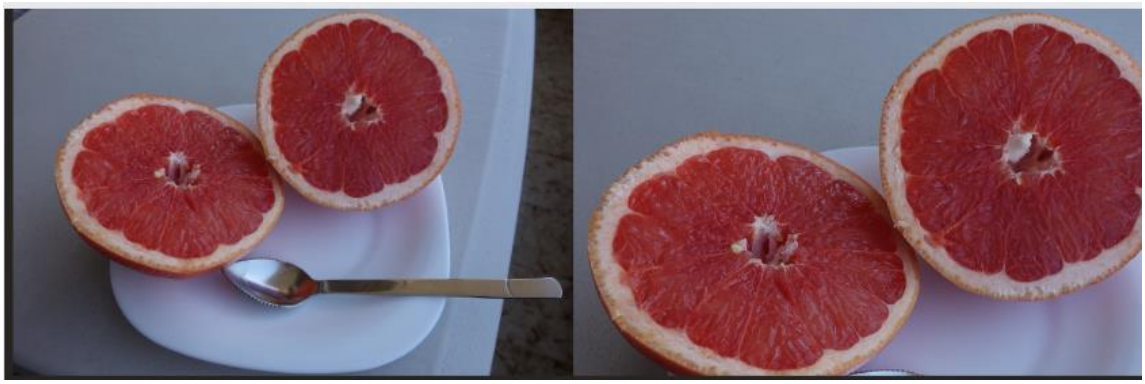
Figura 23 - Baixando classe "*Window*"

```
!python3 downloadOI.py --classes 'Window' --mode train --groupOf=0 --inside=0
```

Fonte: Própria autora

Após isso, para destacar apenas a classe desejada nas imagens, foram realizados recortes utilizando a ferramenta de edição de imagem nativa do sistema operacional *Windows*. Esses recortes foram feitos de forma cuidadosa para garantir que apenas a classe de interesse estivesse presente nas imagens resultantes. Na Figura 24 é apresentado o antes e depois desse tratamento.

Figura 24 - Recorte nas imagens

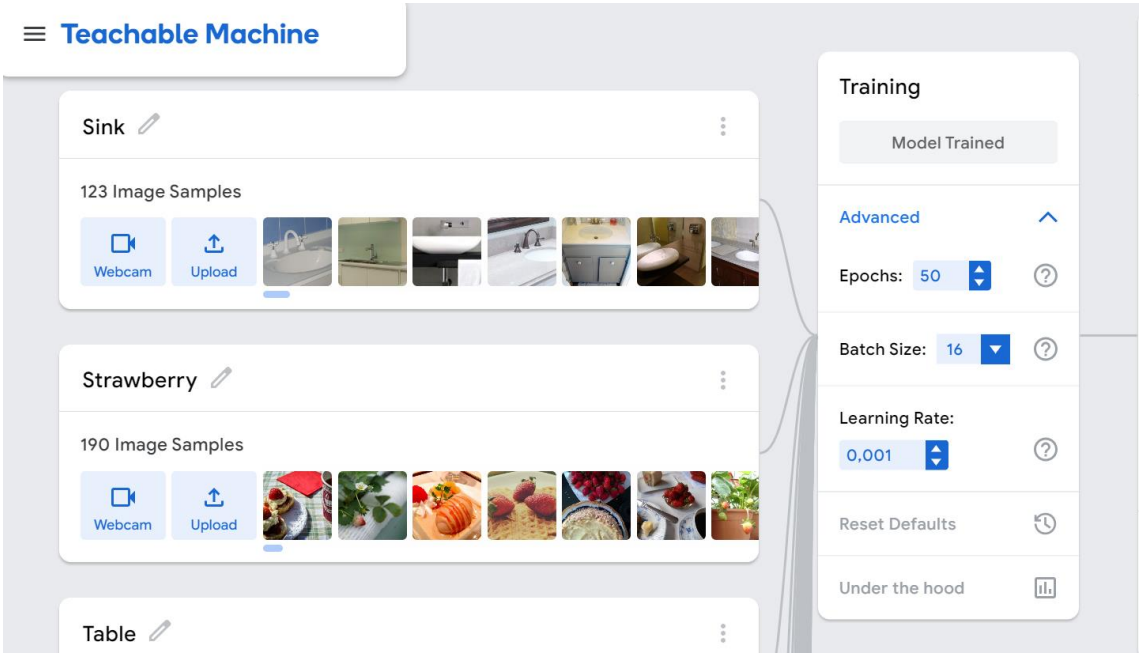


Fonte: Própria autora

6.2.2 Treinamento do modelo de classificação

Com as imagens devidamente tratadas foi feita a implementação do modelo, Na Figura 25 mostra o ambiente da ferramenta *Teachable Machine,* então ao colocar as imagens, são definidos parâmetros iniciais que foram explicados anteriormente neste trabalho no tópico do *Teachable Machine*. As configurações iniciais são 50 *epochs,* 16 *batch size* e 0,01 de *learning rate.*

Figura 25 - ambiente *Teachable Machine*



Fonte: Teachable Machine (2022)

Diante disso foi feito o treino com esses parâmetros iniciais para ver como o modelo se comportaria. Assim, Na Tabela 11 contém o resultado da acurácia por classe, que segundo *Teachable Machine* (2022) representa o total de acertos em relação ao total de previsões realizadas, nela algumas classes apresentam uma acurácia mais baixa, como "*Towel*" com 11% e " *Washing_machine* " com 33%, indicando que o modelo tem dificuldades em identificar corretamente essas classes. Isso pode ser devido à semelhança visual entre essas classes e outras classes, o que torna a classificação mais desafiadora. Porém algumas classes obtiveram uma acurácia muito alta, como "*Popcorn*" e "*Calculator*", ambas com 100%. Isso indica que o modelo é muito preciso na classificação dessas classes.

Tabela 11 - Acurácia por classe (modelo com 50 épocas)

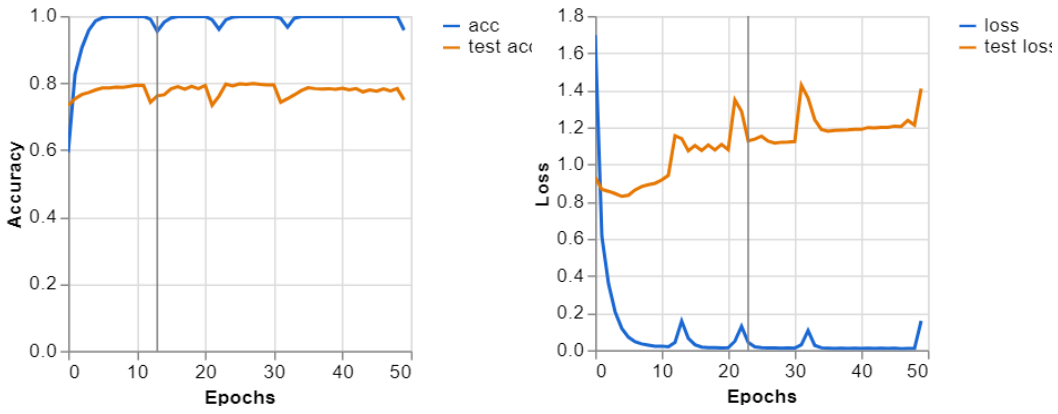
Classe	Acurácia	Qtd de Amostras
Sink	63%	19
Strawberry	86%	22
Table	56%	16
Television	87%	23
Toothbrush	71%	17
Spoon	50%	14
Snake	71%	38
Towel	11%	9
Tree	75%	8
Umbrella	75%	9
Washing_machine	33%	8
Window	71%	17
Whiteboard	75%	16
Watermelon	55%	22
Scissors	75%	20
Refrigerator	88%	25
Vase	67%	18
Tomato	92%	26
Truck	91%	22
Stapler	50%	6
Rabbit	82%	11
Potato	72%	18
Popcorn	100%	6
Plate	75%	20
Pillow	42%	19
Pineapple	55%	11
Pig	77%	22
Person	65%	20
Pencil_sharpener	71%	14
Pencil_case	64%	11
Pen	62%	21
Remote_control	75%	8
Mug	94%	17
Mouse	45%	11
Motorcycle	77%	13
Monkey	100%	20
Microwave	75%	16
Lizard	100%	21
Lemon	56%	9
Hand	83%	6
Foot	82%	11
Laptop	88%	17
House	85%	33

Horse	67%	15
Hat	63%	8
Grape	79%	14
Grapefruit	59%	17
Glasses	83%	6
Gas_stove	94%	17
Frog	73%	30
Fish	73%	15
Eraser	45%	11
Elephant	100%	117
Dog	59%	17
Cucumber	67%	9
Couch	73%	8
Computer_mouse	88%	16
Clock	64%	14
Chair	73%	15
Car	100%	17
Cat	75%	16
Carrot	75%	122
Calculator	100%	14
Book	88%	8
Apple	93%	15
Backpack	78%	9

Fonte: *Teachable Machine* (2022)

Já na Figura 26 os resultados da medida de acurácia geral, ela estabilizou após a 33ª época, ficando com acurácia de 73% no teste. Além disso, também na Figura 26, na perda, que segundo *Teachable Machine* (2022) é uma medida para avaliar o quão bem um modelo aprendeu a prever as classificações corretas para um determinado conjunto de amostras. Se as previsões do modelo forem perfeitas, a perda é zero; caso contrário, a perda é maior que zero.

Figura 26 - Acurácia e perda



Fonte: *Teachable Machine* (2022)

Isso significa que, se a taxa de perda estabilizou em um valor maior que 0, pode ser um sinal de *overfitting*, ou seja, o modelo está muito especializado nos dados de treinamento e não consegue generalizar bem para dados novos ou de validação e foi o que aconteceu de acordo com os resultados.

Portanto, diante disso, é válido mudar os parâmetros para obter um resultado melhor, em vista que a taxa de perda se estabilizou na 35ª época, podemos mudar o parâmetro de época para 35 e avaliar novamente o resultado. Desse modo o modelo foi treinado novamente com este parâmetro e na Tabela 12 é visto de novo a acurácia por classe, na tabela é possível observar que após a mudança de parâmetro há mais classes com valores de acurácia superiores a 50%, o que indica que o modelo melhorou o desempenho em detectar essas classes e as classes que obtiveram um resultado baixo anteriormente também melhoraram, "*Towel*" foi para 56% e "*Washing_machine*" para 71%.

Tabela 12 - Acurácia por classe (modelo com 35 épocas)

Classe	Acurácia	Qtd de Amostras
Sink	95%	19
Strawberry	79%	29
Table	38%	16
Television	74%	23
Toothbrush	53%	17
Spoon	71%	14
Snake	82%	38
Towel	56%	17
Tree	75%	29
Umbrella	89%	13
Washing machine	71%	7
Window	100%	17
Whiteboard	88%	16
Watermelon	68%	22
Scissors	80%	20
Refrigerator	84%	25
Vase	56%	18
Tomato	96%	26
Truck	86%	22

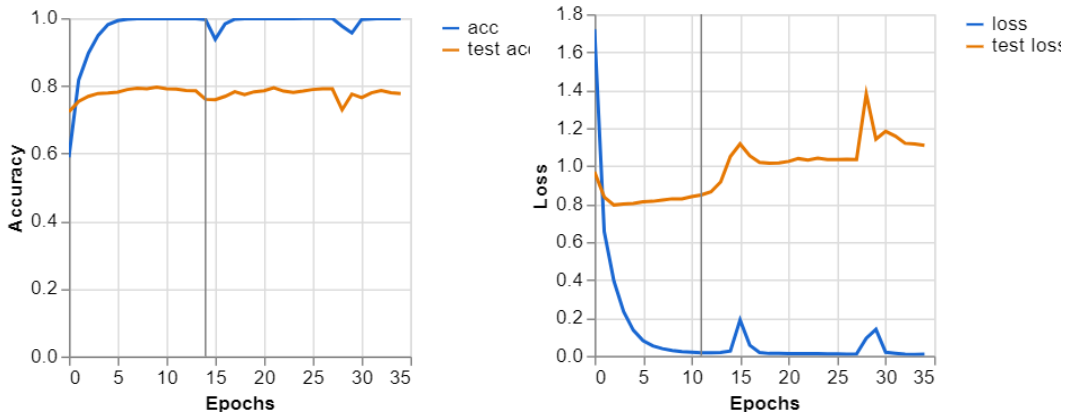
Stapler	83%	6
Rabbit	73%	11
Potato	89%	18
Popcorn	100%	6
Plate	60%	20
Pillow	74%	19
Pineapple	91%	11
Pig	91%	20
Person	90%	20
Pencil sharpener	86%	14
Pencil case	82%	11
Pen	62%	21
Remote_control	88%	8
Mug	100%	17
Mouse	73%	11
Motorcycle	100%	13
Monkey	75%	20
Microwave	81%	16
Lizard	48%	21
Lemon	78%	9
Hand	50%	6
Foot	82%	11
Laptop	88%	17
House	91%	33
Horse	80%	15
Hat	100%	8
Grape	64%	14
Grapefruit	94%	17
Glasses	83%	6
Gas stove	88%	17
Frog	87%	30
Fish	53%	15
Eraser	55%	11
Elephant	100%	117
Dog	59%	17
Cucumber	67%	9
Couch	73%	8
Computer_mouse	88%	16
Clock	64%	14
Chair	73%	15
Car	100%	17
Cat	75%	16
Carrot	75%	122
Calculator	100%	14
Book	88%	8

Apple	93%	15
Backpack	78%	9

Fonte: *Teachable Machine* (2022)

Agora em relação a acurácia, podemos ver na Figura 27 que a acurácia aumentou para 77% no teste, também podemos ver que na perda o modelo anterior no teste tinha chegado a 1.4 e agora diminuiu para 1.1 e não teve uma estabilização nas últimas épocas, o que indica que o *overfiting* foi um pouco controlado, agora o modelo possui uma capacidade maior de generalização.

Figura 27 - Acurácia e perda



Fonte: *Teachable Machine* (2022)

Com base nisso, o modelo está pronto para exportação no formato *.tflite*, a fim tornar o modelo mais leve e rápido (*TENSORFLOW LITE*, 2017).

6.3 Desenvolvimento do aplicativo *EnglishAI*

Nesta etapa, foi realizada a implementação do aplicativo mobile com foco no público-alvo, composto por crianças de 5 a 10 anos. Utilizando a linguagem *dart* e o framework de *frontend flutter*. Nesta seção será explicado as técnicas utilizadas e implementação das funcionalidades.

Ao desenvolver o aplicativo, foi priorizada a criação de um design atraente e cativante para as crianças. Levando em consideração a faixa etária e características cognitivas do público-alvo, cores vibrantes, elementos lúdicos e personagens foram incorporados à interface para despertar o interesse e a interação das crianças durante o aprendizado. Durante o processo de design do aplicativo, foram aplicadas técnicas de UI e UX *design* para garantir uma experiência de usuário fluida e agradável. Através de uma abordagem centrada

no usuário, foram adotados os seguintes princípios com base nas obras de Nielsen (1993):

Simplicidade e Intuitividade: A interface foi projetada de forma a minimizar o número de cliques e tornar a navegação intuitiva para as crianças, facilitando o acesso às funcionalidades do aplicativo.

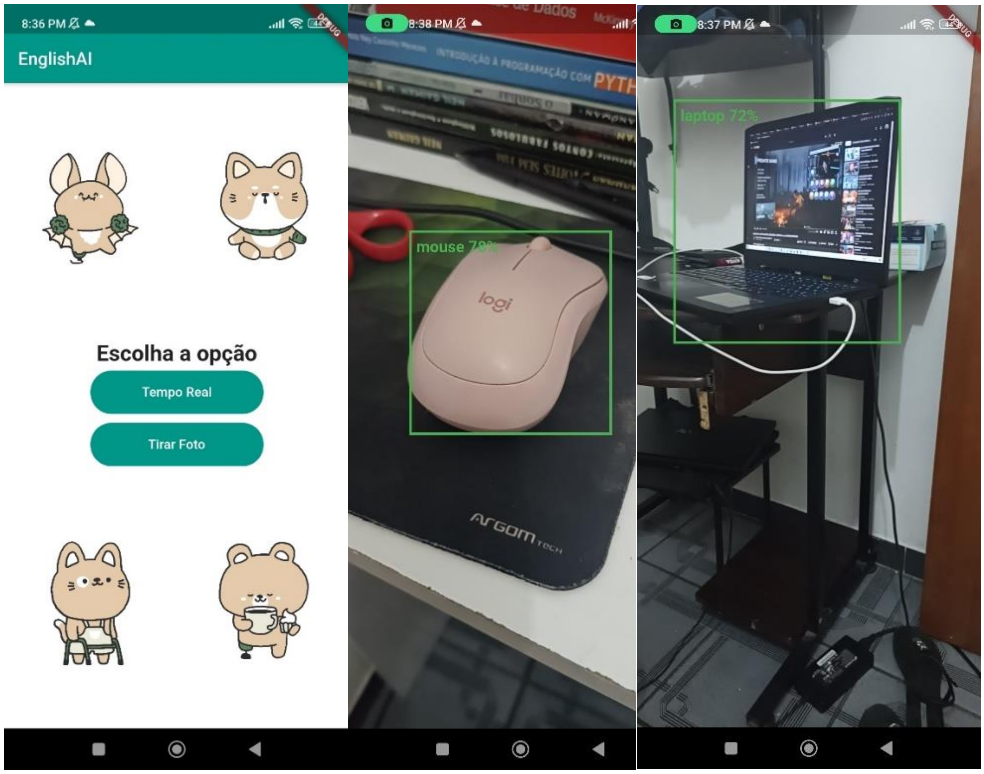
Feedback Visual e Sonoro: Para fornecer um retorno imediato às ações das crianças, foram implementados feedbacks visuais, como animações e transições suaves, bem como feedbacks sonoros, como ao ouvir a pronúncia da palavra.

Consistência Visual: Foi mantida uma consistência visual em todo o aplicativo, com o uso de ícones, botões e elementos visuais coerentes, para proporcionar uma experiência familiar aos usuários.

Acessibilidade: Considerando a importância da inclusão, foram aplicadas práticas de design acessível para garantir que o aplicativo seja utilizável por crianças com diferentes níveis de habilidades e necessidades.

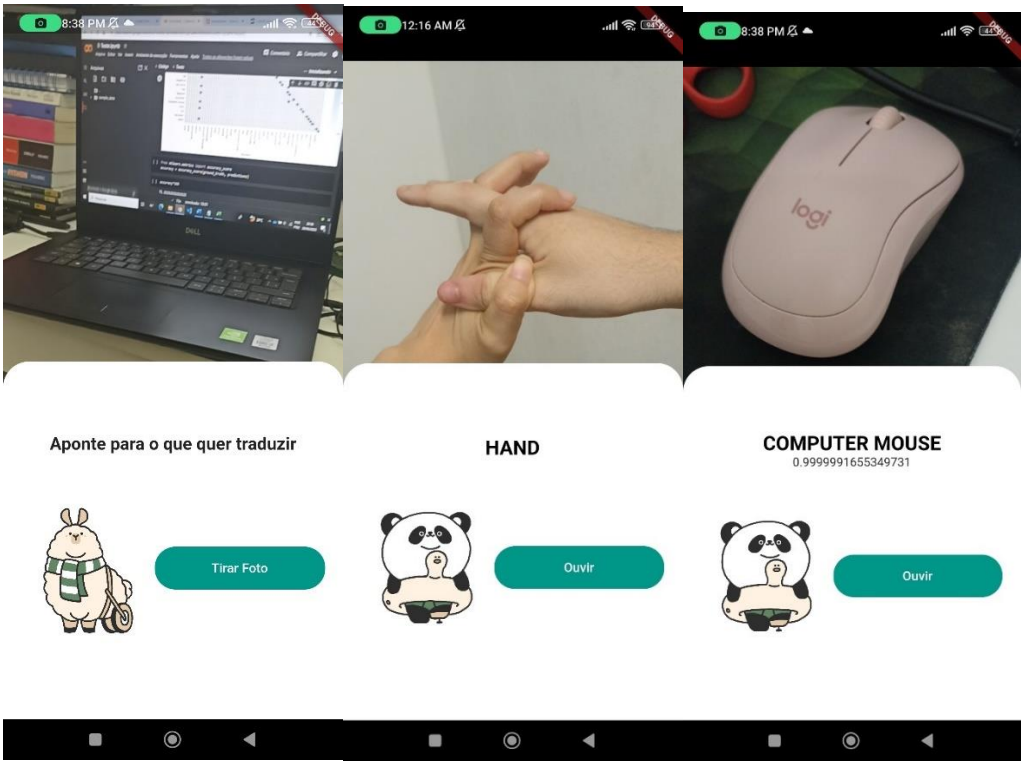
Como dito, o aplicativo foi feito em *dart* com o framework *flutter*, para implementar os modelos foi utilizada a biblioteca *tflite*, onde na mesma há funções para rodar diferentes modelos, como foi apresentado no tópico do diagrama de classes. Na figura 28 contém a tela inicial do app, em que o usuário pode escolher em utilizar a função de detectar em tempo real ou apertar para tirar a foto do elemento, há também a funcionalidade de detecção, em que ao passar a câmera em um objeto que o modelo reconheça é formado na tela uma caixa delimitadora com o nome e porcentagem da classe predita. Já na figura 29 é a funcionalidade de tirar a foto do objeto, em que o usuário deve apontar a câmera e tirar foto do objeto a ser traduzido para inglês, assim também como a tela em que o modelo de classificação retorna a classe que previu em inglês na tela, tendo a funcionalidade de ouvir a pronúncia da palavra em inglês ao apertar o botão de ouvir. Finalizando assim o fluxo da aplicação.

Figura 28 - Telas do *EnglishAI*



Fonte: Própria autora

Figura 29 - Telas do *EnglishAI*



Fonte: Própria autora

CONCLUSÃO

Para execução do presente software propomos a utilização de métodos de classificação de imagem baseados em visão computacional para tornar o aprendizado lúdico e interativo para as crianças. Através de uma aplicação mobile, implementou-se as tecnologias de detecção de objetos e classificação de imagens que permitiu criar um aplicativo envolvente e interativo para elas. Através da API de detecção de objetos do *TensorFlow*, implementou-se um modelo de detecção em tempo real, permitindo que as crianças interajam com objetos do seu ambiente e associem palavras em inglês a esses objetos, além do modelo de classificação de imagens com treinamento baseado no *Open Images Dataset v4*. Essa abordagem permitiu que o aplicativo traduzisse as imagens capturadas pelas crianças em palavras em inglês, palavras essas que foram escolhidas considerando elementos do cotidiano e do ambiente em que as crianças estão inseridas, com isso, mostrando na tela sua pronúncia, assim proporcionando um aprendizado lúdico e eficaz que pode auxiliar no desenvolvimento do inglês nas séries iniciais do Ensino Fundamental.

Este trabalho abre caminho para futuras pesquisas e melhorias, como a expansão do conjunto de dados, refinamento do modelo e incorporação de técnicas adicionais de inteligência artificial, a fim de aprimorar ainda mais o desempenho e a eficácia do sistema. Espero que esta pesquisa contribua para a educação de crianças na aprendizagem do inglês, proporcionando um ambiente de ensino mais estimulante e acessível.

REFERÊNCIAS

ASSIS, Rosuel & Cossío, Jorge B. & Cunha, André & Pitombo, Cira & Fernandes Jr, José. **Uso de Redes Neurais Artificiais para o desenvolvimento de modelos de previsão da condição de pavimentos de aeroportos**. 2016. Disponível em: < https://www.researchgate.net/publication/305463781_Uso_de_Redes_Neurais_Artificiais_para_o_desenvolvimento_de_modelos_de_previsao_da_condicao_de_pavimentos_de_aeroportos> Acesso em: 07 jul. 2023

BERGER, C. **Perceptrons - The Most Basic Form of Neural Network**. 2016. Disponível em: <https://appliedgo.net/perceptron/> Acesso em: 07 jul. 2023

BOENTE, Alfredo; BRAGA, Gláucia. **Metodologia científica contemporânea**. Rio de Janeiro: Brasport, 2004.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer vision with the OpenCV library**, 2008.

CACCIA, L. C. M. **Réseaux de neurones artificiels**. 2018. Disponível em: <http://accromath.uqam.ca/2018/09/reseaux-de-neurones-artificiels/> Acesso em: 07 jul. 2023.

CAPPELI, E. **Desenvolvimento Híbrido com Flutter: Prós e Contras**, 2018. Disponível em: <https://medium.com/@devmob/desenvolvimento-h%C3%ADbrido-com-flutter-pr%C3%B3s-e-contras-6f3f422c480c > Acesso em: 11 jul. 2023.

CRYSTAL, D. (2003). **English as a Global Language**. Cambridge University Press.

DART, **Dart,** 2011. Disponível em: < https://dart.dev/ > Acesso em: 11 jul. 2023.

DATAGEN, 2011**. MS COCO Dataset: Using it in Your Computer Vision Projects.** Disponível em: <https://datagen.tech/guides/image-datasets/ms-coco-dataset-using-it-in-your-computer-vision-projects/#> Acesso em: 11 jul. 2023.

Data Science Academy. **Deep Learning Book**, 2022. Disponível em: <https://www.deeplearningbook.com.br/> Acesso em: 07 jul. 2023.

DONDI, Claudio.; MORETTI, Michela. **A methodological proposal for learning games selection and quality assessment**. 2007. Disponível em:< https://bera-journals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8535.2007.00713.x>. Acesso em: 27 mai. 2022.

EISHIMA, Rubens. **Volume de downloads do Google Play é três vezes maior que o da App Store,** 2020. Disponível em: <https://canaltech.com.br/apps/volume-de-downloads-do-google-play-e-tres-vezes-maior-que-o-da-app-store-173056/ > Acesso em: 14 ago. 2022.

ELFATIMI, E.; ERYIGIT, R.; ELFATIMI, L. **Beans Leaf Diseases Classification Using MobileNet Models**. 2022.

FLUTTER, **Flutter,** 2015. Disponível em: <https://flutter.dev/> Acesso em: 11 jul. 2023.

GEITGEY, A. **Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks**. 2016. Disponível em: <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721> Acesso em: 08 jul. 2023.

GOODFELLOW, I., Bengio, Y., & Courville, A. (2016). **Deep Learning**. MIT Press.

JOBU, **Rede Neural Convolucional (CNN),** 2023. Disponível em: <https://jobu.com.br/2020/08/31/rede-neural-convolucional-cnn/> Acesso em: 08. jul. 2023.

KRUCHTEN, Philippe. **The Rational Unified Process: An Introduction**, 2004.

KUZNETSOVA, A. 2018. **The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale.** Disponível em: < https://arxiv.org/abs/1811.00982> Acesso em: 11 jul. 2023.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. **SSD: Single Shot MultiBox Detector**. Disponível em: <https://arxiv.org/abs/1512.02325>. Acesso em: 10 jul. 2023.

MAZUR, M. A **Step by Step Backpropagation Example**. 2015. Disponível em: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/> Acesso em: 08 jul. 2023.

MURZOVA, A. **Learn OpenCV.** 2020. Disponível em: <https://learnopencv.com/image-classification-with-opencv-for-android/> Acesso em: 09 jul. 2023.

NAYAK, S. **Fast Image Downloader for Open Images V4,** 2019. Disponível em: < https://learnopencv.com/fast-image-downloader-for-open-images-v4/> Acesso em: 14 jul. 2023.

NIELSEN, J; MACK, R.L**. Usability Inspection Methods.** New York: John Wiley & Sons, 1994.

NORMAN, Donald A**. Design emocional: por que adoramos (ou detestamos) os objetos do dia-a-dia**. Rio de Janeiro: Rocco, 2008.

OSÓRIO, Fernando Santos, BITTENCOURT João Ricardo. **Sistemas inteligentes baseados em Redes Neurais Artificiais aplicados ao Processamento de Imagens**, 2000. Disponível em: < https://www.researchgate.net/publication/228588719_Sistemas_Inteligentes_baseados

_em_redes_neurais_artificiais_aplicados_ao_processamento_de_imagens>Acesso em: 05 ago. 2022.

PATTANAYAK, S. **Pro Deep Learning with TensorFlow**, 2017.

PRINCE, S. J. **Computer vision: models, learning, and inference**, 2012.

SINGHAL, M. **Object Detection using SSD Mobilenet and Tensorflow Object Detection API : Can detect any single class from coco dataset.** 2020. Disponível em: <https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd0691> Acesso em: 10 jul. 2023.

SALGADO, Danielle. **Pesquisa sobre aplicativos no Brasil: apps mais populares, hábitos e preferências dos brasileiros.** 2022. Disponível em: < https://blog.opinionbox.com/pesquisa-sobre-aplicativos-no-brasil/>. Acesso em: 10 jun. 2022.

SANDLER, M. **MobileNetV2: Inverted Residuals and Linear Bottlenecks**. 2018. Disponível em: < https://arxiv.org/abs/1801.04381v3 > Acesso em: 10 jul. 2023

STANCIU, M. **Teachable Machine,** 2022. Disponível em: < https://brasil.hackclub.com/workshops/teachable-machine/#:~:text=O%20Teachable%20Machine%20%C3%A9%20um,aprendizado%20de%20m%C3%A1quina%20sem%20programa%C3%A7%C3%A3o.> Acesso em: 11 jul. 2023.

Teachable Machine, **Teachable Machine**, 2018. Disponível em: <https://teachablemachine.withgoogle.com/> Acesso em: 17 dez. 2022.

TENSORFLOW. **TensorFlow: An open-source machine learning framework for everyone**. Disponível em: <https://www.tensorflow.org/>. Acesso em: 10 jul. 2023.

TENSORFLOW LITE**. TensorFlow Lite.** Disponível em: <https://www.tensorflow.org/lite>. Acesso em: 10 jul. 2023.

VYGOTSKY, Lev. S. **Aprendizagem e desenvolvimento na Idade Escolar. In: Linguagem, desenvolvimento e aprendizagem**. Vigostky, L. Luria, A. Leontiev, A.N. 11ª. Edição. São Paulo: Ícone, 2010

VYGOTSKY, L.S; LURIA, A.R. & LEONTIEV, A.N. **Linguagem, desenvolvimento e aprendizagem**. São Paulo: Ícone: Editora da Universidade de São Paulo, 1998.