INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE PERNAMBUCO – *CAMPUS* RECIFE

DEPARTAMENTO ACADÊMICO DE CURSOS SUPERIORES

CURSO TECNOLÓGICO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LUIZ EDUARDO PESSOA DE FREITAS

APLICATIVO MÓVEL PARA A IDENTIFICAÇÃO AUTOMÁTICA DOS PRINCIPAIS VETORES DA DOENÇA DE CHAGAS NO ESTADO DE PERNAMBUCO

Recife – PE

2025

LUIZ EDUARDO PESSOA DE FREITAS

APLICATIVO MOBILE PARA IDENTIFICAÇÃO AUTOMÁTICA DOS PRINCIPAIS VETORES DA DOENÇA DE CHAGAS NO ESTADO DE PERNAMBUCO

Trabalho de Conclusão de Curso – TCC apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco como requisito parcial para a obtenção do título de Tecnólogo de Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Ramide Augusto Sales Dantas

Coorientadora: Profª. Drª. Maria Beatriz Araújo Silva

Recife – PE

2025

F866a
2025 Freitas, Luiz Eduardo Pessoa de.
Aplicativo móvel para a identificação automática dos principais vetores da doença de chagas no Estado de Pernambuco / Luiz Eduardo Pessoa de Freitas. --- Recife: O autor, 2025.
56f. il. Color.

TCC (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sitemas) – Instituto Federal de Pernambuco, 2025.

Inclui Referências e apêndices.

Orientador: Professor Dr. Ramide Augusto Sales Dantas

1. Desenvolvimento de Sistema. 2. Aplicativo móvel. 3. Doença de chagas. I. Título. II. DANTAS, Ramide Augusto Sales (orientador). III. Instituto Federal de Pernambuco.

CDD 004.21 (23.ed.)

Catalogação na fonte: Emmely Silva CRB4/1876

LUIZ EDUARDO PESSOA DE FREITAS

APLICATIVO MOBILE PARA IDENTIFICAÇÃO AUTOMÁTICA DOS PRINCIPAIS VETORES DA DOENÇA DE CHAGAS NO ESTADO DE PERNAMBUCO

Trabalho de Conclusão de Curso – TCC apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco como requisito parcial para a obtenção do título de Tecnólogo de Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Ramide Augusto Sales Dantas

Coorientadora: Profª. Drª. Maria Beatriz Araújo Silva

BANCA EXAMINADORA

.

Examinadora interna: Profª. Mª. Renata Freire de Paiva Neves

Instituto Federal de Pernambuco – IFPE – *Campus* Recife

.

Examinadora externa: Profª. Drª. Patrícia Takako Endo

Universidade de Pernambuco – UPE – *Campus* Caruaru

.

Orientador: Prof. Dr. Ramide Augusto Sales Dantas

Instituto Federal de Pernambuco – IFPE – *Campus* Recife

RESUMO

A doença de Chagas é considerada pela Organização Mundial da Saúde uma doença negligenciada e um grave problema de saúde pública, com aproximadamente 7 milhões de pessoas atualmente infectadas pelo Trypanosoma cruzi no mundo. O objetivo deste trabalho foi desenvolver um aplicativo para a plataforma Android que utiliza um modelo de aprendizado de máquina para ajudar a identificar imagens de triatomíneos que são vetores da doença de Chagas. O modelo utilizado foi baseado no EfficientNetV2, uma arquitetura de rede neural convolucional previamente treinada para reconhecimento genérico de imagens, aprimorada com imagens de vetores nativos de Pernambuco e de insetos semelhantes que não são transmissores. O modelo criado foi avaliado usando um conjunto de imagens de teste distintas das usadas em treinamento e validação, apresentando uma acurácia de 91,89%. Esses resultados indicam que o modelo *EfficientNetV2* adaptado conseguiu generalizar bem para novos dados. O aplicativo, denominado TriatoDetect, foi desenvolvido para ter uma interface que facilitasse a identificação de triatomíneos por meio da câmera do celular ou imagens armazenadas no dispositivo.

Palavras-chave: Doença de Chagas; Triatominae; Inteligência Artificial; Aprendizado de Máquina.

ABSTRACT

Chagas disease is considered by the World Health Organization as a neglected disease and a serious public health problem, with approximately 7 million people currently infected by *Trypanosoma cruzi* worldwide. The objective of this work was to develop an application for the Android platform that uses a machine learning model to help identify images of triatomines that are vectors of the Chagas disease. The model used was based on *EfficientNetV2*, a convolutional neural network architecture pre-trained for generic image recognition, which was enhanced with images of native vectors from Pernambuco and similar non-vector insects. The developed model was evaluated using a test dataset distinct from those used in training and validation, achieving an accuracy of 91.89%. These results indicate that the adapted *EfficientNetV2* model was able to generalize well to new data. The application, named TriatoDetect, was designed with an interface to facilitate the identification of triatomines through the phone's camera or images stored on the device.

Keywords: Chagas Disease; Triatominae; Artificial Intelligence; Machine Learning.

SUMÁRIO

1 INTRODUÇÃO

A doença de Chagas é considerada pela Organização Mundial de Saúde (OMS) uma doença negligenciada, apresentando-se como um grave problema de saúde pública, visto que aproximadamente 7 milhões de pessoas estão infectadas pelo *Trypanossoma cruzi* (*T. cruzi*), no mundo (WHO, 2018). Além disso, mantém o padrão epidemiológico de endemicidade em 21 países da região da América Latina e nos Estados Unidos da América, com aproximadamente 70 milhões de pessoas sob risco de exposição à infecção por *T. cruzi* (WHO, 2022).

No Brasil, atualmente, estima-se que cerca de 1,4 a 3,2 milhões de pessoas estejam infectadas pelo *T. cruzi*, e 21,8 milhões estão sob risco de infecção (Souza et al., 2023). Entre os anos de 2010 e 2020 foram registrados 49.574 óbitos, dos quais 11.210 ocorreram na região Nordeste, sendo 1.299 em Pernambuco (Brasil, 2022). O surto mais recente da doença de chagas no Estado, ocorreu em 2019 no município de Ibimirim-PE, ao qual supostamente foi associado à ingestão de alimentos contaminados, resultando em aproximadamente 30 pessoas infectadas (Jansen et al., 2020).

O agente etiológico da doença de Chagas é o protozoário *T. cruzi*, sendo o triatomíneo, inseto hematófago, popularmente conhecido como barbeiro, o principal vetor dessa doença. Esses vetores estão amplamente distribuídos pelo território nacional, com registros de 66 espécies no Brasil (Silva, 2022) e 14 no estado de Pernambuco, destacando como principais de interesse à saúde pública as espécies *Panstrongylus megistus*, *Panstrongylus lutzi*, *Triatoma brasiliensis* e *Triatoma pseudomaculata* (Silva et al., 2021; Medeiros et al., 2023). Assim sendo, Pernambuco é considerado uma região de risco para transmissão vetorial, visto que há registros de triatomíneos em mais de 90% dos municípios pernambucanos (Silva et al., 2012).

Entretanto, para identificação e diferenciação das diferentes espécies de triatomíneos de importância epidemiológica, além de outros insetos semelhantes, faz-se necessário conhecimento acerca da entomologia, visto que a família *Reduviidae* é constituída por 25 subfamílias, entre elas a subfamília Triatominae, as quais as espécies vetores da doença de Chagas estão alocadas (Menezes, 2018).

Por essa razão, estas subfamílias apresentam características morfológicas semelhantes entre si que complicam o processo de identificação dos triatomíneos, podendo levar a erros,

especialmente entre a população em geral e entre profissionais que atuam nessa área, como os Agentes de Endemias, que podem não ter conhecimento suficiente das diferenças morfológicas.

Contudo, ressalta-se a importância da educação e da participação popular na notificação de triatomíneos, seja por meio de vigilância ativa ou outras formas de engajamento, visando contribuir para que as gerências de saúde possam implementar ações de controle nas áreas de maior vulnerabilidade e risco de maneira mais eficiente, buscando assim reduzir a incidência da doença de Chagas (Silva et al., 2022).

Perante o exposto, buscando inserir a população no processo de identificação dos insetos transmissores da doença de Chagas e garantir a notificação aos setores responsáveis em tempo real, sugere-se o desenvolvimento de um aplicativo móvel para o reconhecimento dos triatomíneos, além de diferenciar dos insetos que são morfologicamente semelhantes. Podendo assim, esse aplicativo ser utilizado pelos profissionais de saúde e pela população.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho foi desenvolver um aplicativo móvel para a plataforma Android capaz de identificar automaticamente e registrar geograficamente, por meio de técnicas de *Deep Learning* e georreferenciamento, imagens de triatomíneos transmissores da doença de Chagas, distinguindo-os de insetos semelhantes não vetores, dessa forma auxiliando a população em geral na prevenção da doença e as autoridades sanitárias no mapeamento de áreas críticas.

1.1.2 Objetivos Específicos

- Configurar e treinar o modelo de *Deep Learning* para detecção dos triatomíneos nas imagens;
- Desenvolver um aplicativo móvel que usa o modelo desenvolvido para identificar os triatomíneos;

2 REFERENCIAL TEÓRICO

A doença de Chagas, também conhecida como tripanossomíase americana, é uma enfermidade tropical negligenciada causada pelo protozoário *Trypanosoma cruzi*. Ela é transmitida principalmente por insetos hematófagos conhecidos popularmente como **barbeiros**, pertencentes à subfamília *Triatominae*. Os barbeiros contaminados eliminam o *T. cruzi* em suas fezes ao picar o hospedeiro, permitindo que o protozoário penetre no organismo humano por meio de feridas ou mucosas (WHO, 2022).

Figura 1 - Imagem de *Triatoma brasiliensis*, triatomíneo com ampla distribuição em Pernambuco, disponibilizada pelo Laboratório Nacional e Internacional de Referência em Taxonomia de Triatomíneos, Instituto Oswaldo Cruz – Fundação Oswaldo Cruz (Fiocruz), Rio de Janeiro.

Fonte: Fiocruz, 2023.

A identificação de triatomíneos é desafiadora devido à diversidade de espécies e à sua distribuição irregular em diferentes habitats. Estudos em Uberlândia – Minas Gerais, mostraram que espécies como *Panstrongylus megistus* estão presentes tanto em ambientes silvestres quanto domiciliares, o que indica um alto risco de infestação. Esses resultados destacam a necessidade de abordagens integradas, que combinem análises morfológicas detalhadas e monitoramento contínuo, para identificar corretamente os vetores da doença de Chagas e adotar medidas de controle eficazes (Mendes & Lima, 2008).

Já no estado de Pernambuco, o estudo de Silva et al. (2015) evidenciou que a espécie *T. brasiliensis* foi a mais prevalente nas cinco regiões do estado, com 37,3% das amostras, seguida de *Triatoma pseudomaculata* com 36,1% das amostras de um total de 2.443 insetos coletados. Além dessas, a espécie *Panstrongylus megistu*s foi encontrada em todo o estado em regiões descontínuas, variando desde a região do Agreste ao Sertão, onde o município de Santa Cruz do Capibaribe se destacou com 19 (0,57%) registros da espécie. A espécie *Panstrongylus lutzi*,

com 93 (2,7%) ocorrências, teve presença marcante no município de Caruaru, também localizado no Agreste.

No livro Vetores da doença de Chagas no Brasil, Galvão et al. (2014), apresenta os principais triatomíneos em cada estado brasileiro e as principais características morfológicas, destacando as seguintes espécies de vetores da doença de Chagas em Pernambuco: *Panstrongylus lutzi, Panstrongylus megistus, Psammolestes tertius, Rhodnius. nasutus, Rhodnius neglectus, Triatoma brasiliensis, Triatoma pseudomaculata, Triatoma melanocephala, Triatoma petrocchiae, Triatoma rubrofasciata, Triatoma sordida e Triatoma tibiamaculata.*

As ações humanas de expansão e interferência nos habitats desses vetores, bem como as intervenções para controle de alguns dos principais vetores da doença de Chagas (como as ações de erradicação domiciliar do *Triatoma infestans*) acarretam na mudança ou substituição por outros vetores com potencial para domiciliar-se. Como identificado por Silva et al. (2019), as espécies *Triatoma brasiliensis* e *Triatoma pseudomaculata* passaram a se destacar como principais vetores da doença de Chagas no estado, sendo considerados triatomíneos nativos do Nordeste que infestam tanto o intradomicílio quanto o peridomicílio. Outra espécie que vem se destacando nas coletas de campo é a *Panstrongylus lutzi*, que apresenta proporcionalmente uma taxa de infecção natural superior às demais espécies encontradas.

No estudo de Silva et al. (2021), foram coletados no período de 2012 – 2017, 9.738 espécimes de triatomíneos pertencentes a seis espéciesEntre eles, destacaram-se: *Triatoma brasiliensis*, com 8.251 exemplares, *Triatoma pseudomaculata*, com 1.323 exemplares, *Panstrongylus lutzi*, com 100 exemplares, *Triatoma sordida*, com 56 exemplares, *Panstrongylus megistus*, com 7 exemplares e *Rhodnius neglectus*, com 1 exemplar. As espécies *P. lutzi, T. brasiliensis* e *T. pseudomaculata* foram encontradas em todos os municípios do Sertão de São Francisco. A taxa de infecção para flagelados morfologicamente semelhantes a *T. cruzi* nos triatomíneos examinados foi de 2%. Das seis espécies coletadas, quatro foram encontradas positivas: *P. lutzi, T. brasiliensis, T. pseudomaculata* e *T. sórdida* (Silva et al., 2021). Corroborando Silva et al. (2021), o trabalho de Medeiros et al. (2023) demonstrou que as espécies de triatomíneos *T. brasiliensis*, *T. pseudomaculata* e *P. lutzi* estavam amplamente distribuídas no estado de Pernambuco se comparado com outras espécies. Além disso, as espécies *P. lutzi* e *P. megistus* apresentaram maior taxa de infectividade com provável *T. cruzi.*

Ademais, dentre os principais desafios para controle desses vetores da doença de Chagas, temos a presença de insetos semelhantes morfologicamente aos triatomíneos, como demonstrado no estudo de Menezes (2018). Tem-se como exemplo de insetos semelhantes aqueles da subfamília *Ectrichodiinae*, que inclui hemípteros e entomófagos, que se alimentam da linfa de outros insetos, sendo predadores e não apresentando hábitos hematófagos, e por essa razão, não representam uma ameaça à saúde humana. No entanto, devido a semelhanças morfológicas, podem ser confundidos com triatomíneos pelo público leigo, como apresentado na **Figura *2***, onde é perceptível as semelhanças entre o *Triatoma pseudomaculata*, inseto transmissor do *Trypanosoma cruzi*, com o inseto da família *Reduviidae* e subfamília *Ectrichodiina,* o *Pothea jaguaris*, um inseto não transmissor. Da mesma forma, algumas famílias de *Hemiptera* fitófagos, como a *Coreidae*, que se alimentam da seiva das plantas, também são frequentemente confundidas com triatomíneos devido à sua aparência similar (Menezes, 2018).

Figura 2 - Exemplo de inseto morfologicamente semelhante ao Triatomíneo. À esquerda tem-se o *Triatoma pseudomaculata*, inseto transmissor do *T. cruzi*; à direita, o *Pothea jaguaris*, inseto não transmissor.

Fonte: BioDiversity4All.org (2024)

Dado o desafio de identificar corretamente os transmissores da doença de Chagas, Parsons et al. (2020) propôs o uso de técnicas de visão computacional para realizar a identificação automática dos triatomíneos. Esse trabalho utilizou a análise de componentes principais (PCA) para extração de características e aplicou as técnicas *Support Vector Machine* (SVM) e *Random Forest* (RF) na fase de classificação. O método PCA-SVM obteve uma acurácia de 87,62% para 410 imagens de 12 espécies mexicanas e 75,26% para 1.620 imagens de 39 espécies brasileiras. Já o método PCA-RF alcançou 100% de precisão para ambas as espécies. Este resultado, no entanto, indica um possível *overfitting* do modelo ao conjunto de

dados utilizado, sugerindo baixa capacidade de generalização do modelo para uso em cenários reais. Esse aspecto reforça a escolha por arquiteturas modernas de *Deep Learning*, que apresentam maior robustez para lidar com variabilidade de imagens em ambientes não controlados.

3 METODOLOGIA E DESENVOLVIMENTO

3.1 Desenvolvimento do Modelo

3.1.1 Seleção dos Conjuntos de Imagens

Após a identificação dos triatomíneos de interesse epidemiológico em Pernambuco (Capítulo 2), foi realizada a busca por imagens dos insetos para treinamento e teste do modelo. As imagens dos insetos transmissores foram obtidas primeiramente do conjunto de dados utilizado por Gurgel-Gonçalves et al. (2017) – trabalho que faz a classificação de vetores da doença de Chagas no México e no Brasil, com 12 espécies mexicanas e 38 brasileiras. Esse trabalho envolveu a automação completa do processamento de imagens antes da análise, transformando-as em um conjunto de marcadores e as distâncias entre eles, e, em seguida, realizando a análise utilizando classificadores estatísticos tradicionais, como análise discriminante linear e redes neurais artificiais. Optou-se por essas imagens por sua disponibilidade gratuita, alta qualidade e uniformidade. Também foram utilizadas fotos da plataforma BioDiversity4all[1] (associação sem fins lucrativos que disponibiliza registros de fauna validados por curadores). Essa complementação foi necessária porque as imagens de Gurgel-Gonçalves et al. (2017) apresentavam características visuais muito semelhantes entre si. Em relação as imagens dos insetos não transmissores que podem ser confundidos com os insetos transmissores da família *Triatoma*, foram obtidas exclusivamente da plataforma BioDiversity4all, devido à dificuldade de encontrar uma base de dados etimológica dessas famílias de insetos. Por fim, a maioria das imagens dos insetos transmissores que compôs a base de treinamento foi obtida dessa mesma plataforma.

Essa diversificação nas fontes de imagens visou a melhoria da qualidade do modelo de identificação de imagens, tornando-o mais robusto na distinção entre diferentes espécies de insetos transmissores e não transmissores. Como as imagens da BioDiversity4all foram registradas por cidadãos em condições reais de captura (com variações de iluminação, ângulos e dispositivos móveis), elas se aproximam do cenário prático de uso do aplicativo, onde os usuários fotografarão insetos em ambientes não controlados.

Por fim, para minimizar a ocorrência de falsos positivos, foram incorporadas imagens do repositório COCO[2], um extenso conjunto de dados utilizado para tarefas de detecção, segmentação e legendagem de objetos. Esse conjunto de imagens foi utilizado para a criação

[1] biodiversity4all.org
[2] cocodataset.org

de uma terceira classe destinada à detecção não identificado, que será acionada quando o sistema não conseguir determinar se o inseto pertence à categoria de transmissor ou não transmissor. A adição de uma classe não identificado segue a abordagem discutida por Nagahama et al. (2023), onde o uso desta técnica se mostrou eficaz para lidar com a classe desconhecida, ajudando a melhorar a robustez e a precisão dos modelos de classificação ao prever de forma confiável quando um objeto ou imagem não se enquadra nas classes de interesse. Com isso, há uma redução de falsos positivos e contribui para um sistema de identificação mais confiável (Nagahama et al., 2023).

3.1.2 Tratamento das Imagens

3.1.2.1 Balanceamento dos Dados

Foi implementado um procedimento de amostragem aleatória, conhecido como *Random Undersampling*, que é uma abordagem utilizada para equilibrar a distribuição do conjunto de dados em cada classe, excluindo aleatoriamente os exemplos da classe majoritária (Chaipanha & Kaewwichian, 2022). Esse procedimento foi realizado objetivando mitigar o desequilíbrio na distribuição original de imagens entre as diferentes espécies, além de reduzir o problema de viés da classificação.

3.1.2.2 Aumentando os Dados

Ao utilizar a técnica de balanceamento *Undersampling*, a base de dados ficou reduzida, resultando em 450 imagens distribuídas igualmente entre três classes: 150 imagens de “Insetos transmissores”, 150 de “Insetos não transmissores” e 150 de “Não identificado”. Como se trata de um problema de classificação multiclasse, essa redução poderia comprometer a capacidade de generalização do modelo. Para contornar essa situação, aplicou-se a técnica de *Data Augmentation* com o objetivo de ampliar a quantidade e a diversidade das imagens disponíveis para cada classe e, assim, mitigar o risco de *overfitting*. De acordo com Chollet (2018), o *overfitting* ocorre quando o modelo se ajusta de forma muito específica aos dados de treinamento e perde a capacidade de generalizar para novos exemplos. Portanto, ao aumentar a base de dados e diversificar os exemplos em todas as classes, buscou-se reduzir esse problema e melhorar o desempenho da classificação. Por este motivo, ao treinar o modelo, foi criada uma função para aplicar camadas de augmentação da biblioteca TensorFlow/Keras[3] para gerar novas imagens a partir das imagens existentes, aumentando a diversidade da base de dados e ajudando

[3] tensorflow.org

a reduzir o problema de *overfitting*. As imagens geradas possuem alterações aleatórias em suas propriedades, como rotação, zoom, brilho e etc. A **Tabela 1** apresenta as propriedades que foram alteradas.

Tabela 1 - Alterações aplicadas às imagens para o aumento de dados usando TensorFlow/keras. Recife, 2024.

Propriedades	**Alteração**
Girar	Vertical / Horizontal
Rotação	-36° / +36°
Cisalhamento	-10% / +10%
Zoom	-10% / +10%
Contraste	-10% / +10%
Brilho	-10% / +10%

Fonte: O autor (2025).

Os valores das alterações na **Tabela 1** são usados na geração de novas imagens a partir do conjunto existente. Como exemplo, o valor da rotação está no intervalo entre -36° e +36°, logo um novo conjunto de imagens é gerado em cada "época" (do inglês *Epoch,* período de uma iteração completa sobre o conjunto de dados durante o treinamento do modelo) onde um valor dentro deste intervalo será aplicado às imagens. Os valores das alterações foram escolhidos para que não houvesse uma variação exagerada das propriedades, comprometendo assim a qualidade das imagens geradas, evitando degradação das texturas e dos detalhes das imagens dos triatomíneos em comparação a base de dados original (exemplos na **Figura 3**).

Figura 3 – Exemplos de imagens após passar pelo processo de aumento de dados

Fonte: O autor (2025).

3.1.3 Base de Treinamento Final

Conforme o levantamento acerca das principais espécies de Triatomíneos em Pernambuco (Capítulo 2), as espécies amplamente distribuídas no Estado e que apresentaram alta taxa de infectividade foram: *Triatoma brasiliensis*, *Triatoma pseudomaculata*, *Panstrongylus megistus* e *Panstrongylus lutzi (*Silva et al., 2015; Silva et al., 2019; Silva, et al., 2021; Medeiros et al., 2023). O total de imagens das espécies de triatomíneos brasileiros, disponibilizados no trabalho de Gurgel-Gonçalves et al. (2017), disponível de forma aberta em https://dx.doi.org/10.5061/dryad.br14k, foi de 1.502 imagens, dos quais 237 foram das espécies de interesse ao estudo, sendo elas: 34 da espécie *Panstrongylus Lutzi*, 84 da espécie *Panstrongylus Megistus*, 64 da espécie *Triatoma Brasiliensis* e 55 da espécie *Triatoma Pseudomaculata*. Entretanto, para evitar que o modelo fosse enviesado por exemplos muito semelhantes, optou-se por trabalhar com apenas quatro imagens de cada espécie. A **Erro! Fonte de referência não encontrada.** têm-se exemplos das imagens do conjunto de dados citado.

Figura 4 - Imagens disponibilizadas no artigo de Gurgel-Gonçalves et al. (2017). Da esquerda à direita, tem-se as espécies *P. lutzi, P. megistus, T. brasiliensis* e *T. pseudomaculata*.

Fonte: Gurgel-Gonçalves et al., 2017.

As imagens encontradas na plataforma BioDiversity4All, referentes a cada espécie de triatomíneos em questão, são exemplificadas na **Figura 5**.

Figura 5 - Imagens disponibilizadas na plataforma BioDiversity4All, da esquerda à direita, têm-se as espécies *P. lutzi, T. brasiliensis, P. megistus* e *T. pseudomaculata*.

Fonte: BioDiversity4All.org (2024).

Com isso a inclusão das imagens do BioDiversity4All, o conjunto final de imagens para insetos transmissores da doença de Chagas ficou disposto conforme mostrado na **Tabela 2**

Tabela 2 - Total de imagens de insetos transmissores utilizados para o treinamento.

Espécies	Quantidade de Imagens
Panstrongylus lutzi	38
Panstrongylus megistus	37
Triatoma brasiliensis	38
Triatoma pseudomaculata	37
Total	**150**

Fonte: elaborado pelo autor (2025).

Em relação aos insetos das subfamílias *Ectrichodiinae* e *Harpactorinae*, que não transmitem a doença e podem ser confundidos por pessoas leigas com os vetores da doença de Chagas, foram utilizadas imagens apenas da plataforma BioDiversity4All. Foram escolhidas aleatoriamente 75 imagens de cada subfamília, totalizando 150 imagens no total, conforme **Tabela 3**. Na **Erro! Fonte de referência não encontrada.**, têm-se exemplos das imagens selecionadas de ambas as subfamílias, *Harpactorinae* e *Ectrichodiinae*.

Tabela 3 - Total de imagens obtidas da plataforma BioDiversity4All correspondentes aos insetos não transmissores da doença de Chagas, para treinamento do modelo. Recife, 2024.

Famílias	Quantidade de Imagens
Subfamília Harpactorinae	75
Subfamília Ectrichodiinae	75
Total	**150**

Fonte: O autor (2025).

Figura 6 - Imagens disponibilizadas na plataforma BioDiversity4All. Da esquerda à direita, têm-se as espécies da subfamília *Harpactorinae, Apiomerus lanipes* e *Isyndus heros*, e da subfamília *Ectrichodiinae, Pothea jaguaris* e *Rhiginia cinctiventris*

Fonte: BioDiversity4All.org (2024).

Por fim, foram escolhidas 150 imagens do conjunto de imagens do *dataset* COCO. Essas são imagens aleatórias, incluindo pessoas, animais, paisagens e outras (**Erro! Fonte de referência não encontrada.**), visando evitar falsos positivos, como descrito na seção 3.1.1.

Figura 7 - Exemplos de imagens aleatórias selecionadas a partir do conjunto de dados do COCO2, para treinamento do modelo classificando as imagens como "Não identificado".

Fonte: cocodataset.org (2024).

Ao final, o conjunto total de imagens disponíveis para o treinamento do modelo ficaram dispostas da seguinte forma (**Tabela 4**):

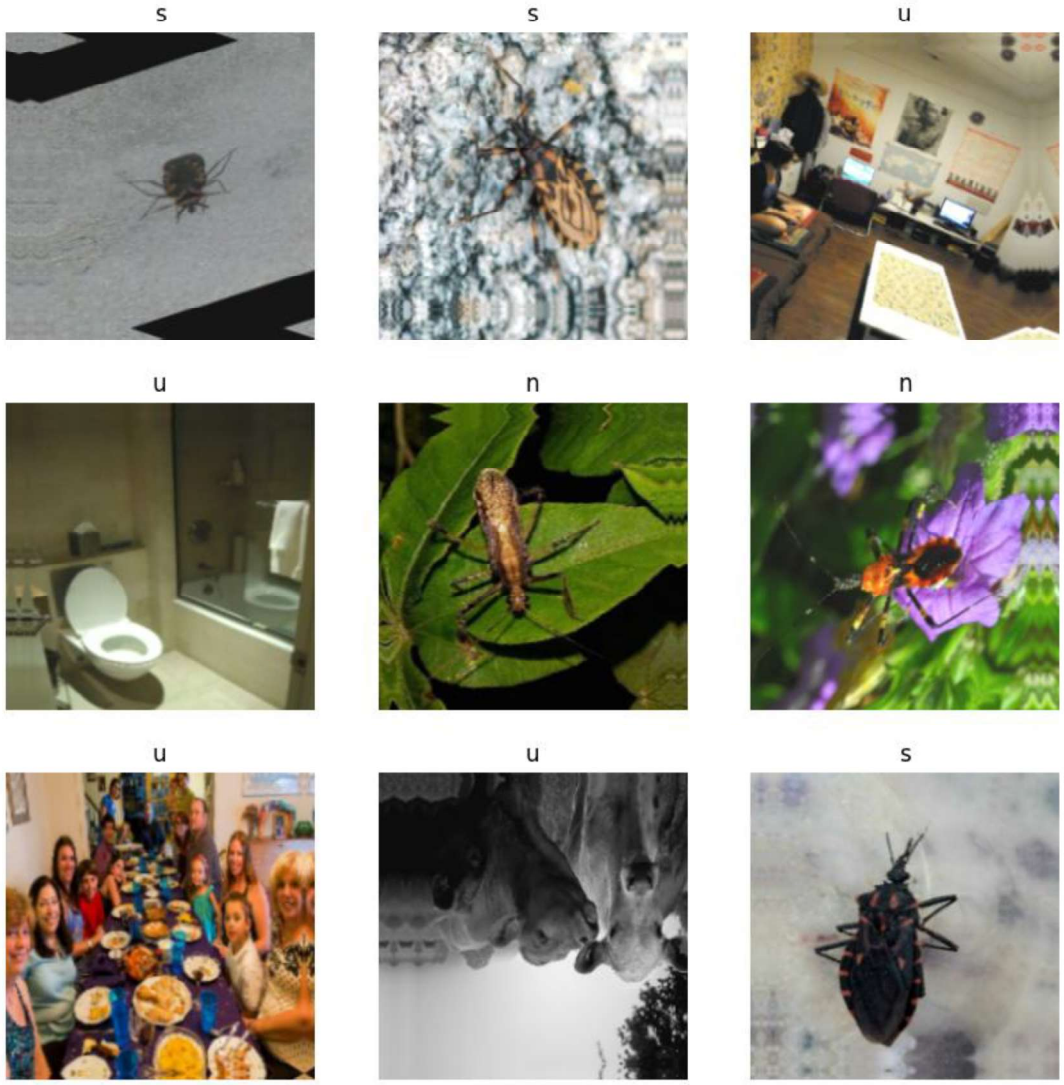
Tabela 4 - Total de imagens utilizadas para treinamento do modelo.

Classificações	Quantidade de Imagens
Insetos transmissores	150
Insetos não transmissores	150
Não identificado	150
Total	**450**

Fonte: elaborado pelo autor (2025).

Sobre o conjunto final foram realizadas as operações descritas na seção 3.1.2.2 objetivando reduzir o *overfitting,* obtendo-se novas imagens a partir da base de dados original, contudo alterando-se os parâmetros. A **Figura 8** exemplifica alguns dos resultados obtidos após a aplicação da função de aumento de dados.

Figura 8 - Imagens obtidas a partir do aumento de dados usando TensorFlow/keras.



Fonte: O autor (2025).

Legenda: n - Inseto não transmissor / s - Inseto transmissor / u - Não identificado

3.1.4 Automação do Treinamento com RoboFlow

Para facilitar o processo de treinamento do modelo, foi utilizada a ferramenta RoboFlow[4], que é uma plataforma que fornece ferramentas para treinar, implantar e gerenciar modelos de visão computacional, como redes neurais convolucionais, para tarefas de processamento de imagens e vídeos. Diante do fluxo de processos que a plataforma RoboFlow[4] é capaz de realizar, foram utilizadas as etapas de "organização" (do inglês - Organize) e "etiquetação" (do inglês - *Label*) dos dados. Mais especificamente, o RoboFlow[4] foi utilizado para agrupar as imagens em cada uma das 3 classes, etiquetar cada imagem com a classe correspondente e por fim, separar o conjunto de imagens em treino, teste e validação (

Tabela 5).

Tabela 5 - Conjuntos de imagens separadas por finalidades. Recife, 2024.

Conjunto	Porcentagem (Qtd. Imagens)
Treino	70% (315 imagens)
Validação	20% (90 imagens)
Teste	10% (45 imagens)

Fonte: O autor (2025).

3.1.5 Escolha, Ajuste e Treinamento do Modelo

Para a implementação do modelo de classificação de imagens proposto neste trabalho, optou-se por utilizar o *EfficientNetV2*, um modelo pré-treinado, como ponto de partida. O *EfficientNetV2* é uma família de arquiteturas de redes neurais convolucionais (CNN) reconhecida por sua notável velocidade de treinamento e eficiência de parâmetros. A escolha deste modelo é corroborada pelo estudo de Devi et al. (2023), que investigou a aplicação do *EfficientNetV2* no reconhecimento de pragas e doenças em plantas. Em sua análise, os autores relataram um bom desempenho do modelo, que alcançou 80,1% de precisão no extenso e desafiador conjunto de dados de pragas IP102, ilustrados na **Figura 9**. O estudo destaca ainda que o modelo atinge essa alta performance a uma velocidade superior em comparação a outras arquiteturas.

[4] roboflow.com

Figura 9 - Amostras de diferentes espécies de insetos do conjunto de dados IP102, um *benchmark* em larga escala para o reconhecimento de pragas agrícolas analisadas em Devi et al. (2023).

Fonte: Devi et al. (2023).

Neste trabalho, o modelo pré-treinado em questão foi ajustado à base de dados seguindo as instruções do tutorial do Keras[5], que abordam etapas como a adaptação das camadas finais do modelo, o ajuste dos hiper parâmetros e o processo de *fine-tuning* para melhor adequação ao novo conjunto de imagens. Ademais, foi utilizada a plataforma Google Colab[6] para conduzir o restante do trabalho, empregando os dados disponíveis no RoboFlow. Um *script* Python, desenvolvido especificamente para este trabalho foi executado no Google Colab, onde o modelo final de classificação dos insetos foi gerado. Esse *script* realiza o processamento dos dados e o treinamento do modelo, e avalia a eficiência e a precisão da classificação. O código completo pode ser consultado no Apêndice A.

3.2 Desenvolvimento do Aplicativo

3.2.1 Desenvolvimento Android Nativo

No desenvolvimento da aplicação, foi utilizado Android nativo com Kotlin[7], que é a linguagem de programação adotada preferencialmente pela plataforma deste 2019 e utilizada por 50% dos desenvolvedores Android[8]. O desenvolvimento nativo possui várias vantagens, dentre as quais, destacam-se que as aplicações nativas possuem um maior desempenho, além do total controle sobre a aplicação na plataforma desejada, melhorando assim significativamente a experiência do usuário (El-Kassas et al., 2017). Além disso, de acordo com o site StatCounter (2024), que oferece estatísticas sobre o tráfego *web*, verificou que, no período de setembro de 2023 à setembro de 2024, a plataforma Android correspondeu a mais de 81,66%

[5] keras.io
[6] colab.research.google.com
[7] kotlinlang.org
[8] kotlinlang.org/android-overview

em participação no mercado de sistemas operacionai móveis no Brasil, evidenciando um elevado uso desse sistema operacional móvel frente aos demais.

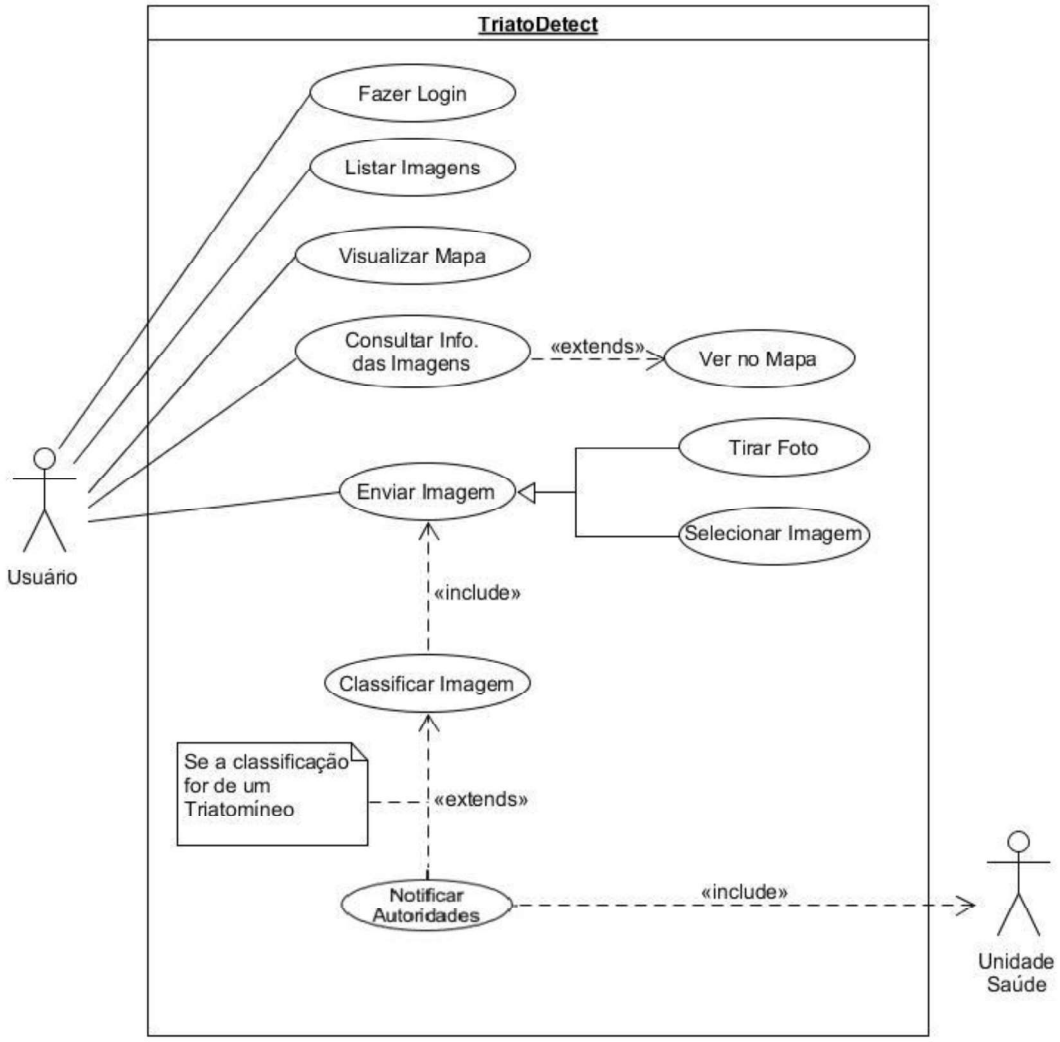
3.2.2 Casos de Uso

Nesta seção são apresentados os principais casos de uso do sistema desenvolvido, o qual foi nomeado de TriatoDetect, sendo a palavra "Triato" escolhida em alusão aos Triatomíneos, insetos vetores da doença de Chagas e a palavra em inglês "*Detect*" que significa "detectar". A seguir são descritos brevemente os casos de uso:

Quadro 1 - Descrição dos Casos de Uso do aplicativo "TriatoDetect". Recife, 2024.

Identificador	Nome do Caso de Uso	Descrição
UC01	Fazer Login	O usuário deve poder autenticar-se no sistema utilizando sua conta Google. O usuário seleciona a opção de login com Google, escolhe a conta do google conectada no dispositivo.
UC02	Listar Imagens	Permite que o usuário visualize a lista de imagens que ele enviou para o aplicativo.
UC03	Consultar Informações das Imagens	Ao clicar um item da lista descrita no UC02, é exibido informações mais detalhadas da classificação, como localização, data e hora.
UC04	Visualizar Mapa	O sistema exibe os pontos geoespaciais em um mapa interativo, permitindo ao usuário visualizar e interagir com todas as classificações, não apenas feitas por ele, mas as classificações de todos os usuários do aplicativo.
UC05	Ver no mapa	A partir da imagem escolhida no UC03, o usuário pode ver no mapa, um pino indicando o ponto exato onde a imagem foi enviada, e ao clicar nesse ponto é exibido o resultado da classificação a data e hora.
UC06	Tirar Foto	O usuário seleciona a opção de tirar foto, o sistema ativa a câmera, e, após a captura, a imagem pode ou não ser cancelada.
UC07	Selecionar Imagem	O usuário acessa a galeria de imagens, escolhe a imagem desejada podendo ou não cancelar a imagem escolhida e selecionar outra.
UC08	Classificar Imagem	O usuário fornece uma imagem ao sistema, que a processa conforme usando o modelo desenvolvido neste trabalho. O modelo classifica a imagem como "Inseto transmissor" ou "Inseto não transmissor" ou "Não identificado".
UC09	Enviar Imagem	Após a classificação da imagem (UC07), o sistema faz o upload da imagem com os metadados, como URL e data de upload, esses dados, junto com a imagem, são armazenados no sistema.
UC10	Notificar Autoridades	Quando a imagem classificada (UC08) é um inseto transmissor da doença de Chagas, um e-mail é enviado para os endereços de e-mails que estão previamente cadastrados.

Figura 10 - Diagrama de Casos de Uso do aplicativo mobile TriatoDetect.



F Fonte: O autor (2024).

3.2.3 Ferramentas utilizadas para o Desenvolvimento de Software

3.2.3.1 Android Studio

O Android Studio[9] é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android. Dentre algumas vantagens de se usar Android Studio pode se destacar suporte a linguagem Kotlin, como também a integração à plataforma de nuvem do Google (*Google Cloud Platform*), facilitando a agregação do aplicativo com os serviços fornecidos pelo ecossistema do Google (Google, n.d).

[9] developer.android.com/studio

3.2.3.2 Firebase

O Firebase é uma plataforma em nuvem do Google que ajuda no desenvolvimento de aplicativos, fornecendo várias funcionalidades já integradas no Android Studio. No aplicativo do presente trabalho, foram usadas as seguintes funcionalidades:

- **Firebase Authentication -** fornece um serviço de *back-end*, SDKs (*Software Development Kit*, um conjunto de ferramentas e bibliotecas que ajuda no desenvolvimento de software para uma plataforma específica) e bibliotecas de IU (Interface de Usuário) prontas para autenticar usuários no seu aplicativo. Esse serviço oferece suporte à autenticação usando senhas, números de telefone, provedores de identidade federados conhecidos, como Google, Facebook e Twitter, entre outros. No aplicativo, a autenticação será feita pelo próprio login fornecido pelo Google.
- **Cloud Storage -** é um serviço de armazenamento de objetos (arquivos), sendo utilizado para salvar as imagens disponibilizadas pelo usuário.
- **Cloud Firestore -** é um banco de dados NoSQL flexível e escalonável para desenvolvimento focado em dispositivos móveis, Web e servidores. Esta funcionalidade foi usada para armazenar os dados das imagens, como a data a qual houve a inserção da imagem, o usuário que a inseriu, o status da classificação, localização, dentro outras informações.
- **Maps SDK for Android -** é uma funcionalidade do Google Cloud que permite a manipulação de mapas por geolocalização dentro do aplicativo. Isso foi necessário para acessar a localização do usuário e com esta informação, anexar nos dados da imagem o local onde a imagem foi adicionada.
- **Cloud Functions -** serviço sem servidor (*serveless*) que permite executar automaticamente o código de *back-end* em resposta a eventos acionados por recursos do Firebase e solicitações HTTPS. Foi criada uma função na nuvem para enviar e-mails às unidades de saúde ou responsáveis pelo monitoramento da doença de chagas no estado de Pernambuco. Esses e-mails são pré-cadastrados no Cloud Firestore do aplicativo.

3.2.3.3 GitHub

O Github[10] é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. O Git[11] é um sistema de controle de versão distribuído, gratuito e de código aberto. A duas tecnologias acima (Git/GitHub) foram utilizadas para o versionamento e repositório do código fonte do aplicativo[12].

[10] github.com
[11] git-scm.com
[12] github.com/TriatoDetect

4 RESULTADOS E DISCUSSÃO

4.1 Resultados do Modelo

Após treinar o modelo, os resultados obtidos foram promissores, apresentando uma perda (*loss*) de 0.0480 e uma acurácia de 98,59% no conjunto de treinamento. A perda é uma medida numérica que quantifica o erro do modelo, ou seja, o quão distante as previsões do modelo estão dos valores corretos. No nosso caso, foi utilizada a função de perda *Cross-Entropy* (Entropia Cruzada), muito comum em tarefas de classificação, que é calculada da seguinte forma:

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \cdot \log(\hat{y}_{i,c})$$

Onde:

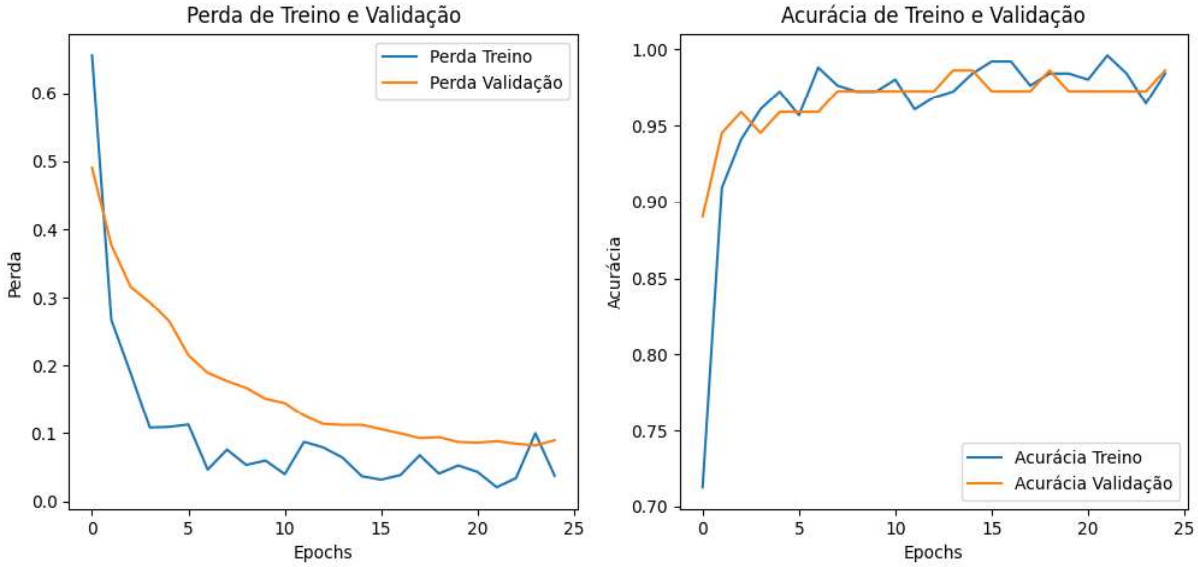
- N é o número total de amostras;
- C é o número de classes;
- $y_{i,c}$ é 1 se a amostra i pertence à classe c e 0 caso contrário;
- $\hat{y}_{i,c}$ é a probabilidade prevista pelo modelo para a classe c.

A **acurácia**, por outro lado, mede a proporção de previsões corretas em relação ao total de previsões feitas pelo modelo, e é calculada como:

$$Acurácia = \frac{Número\ de\ Previsões\ Corretas}{Número\ Total\ de\ Previsões}$$

No conjunto de validação, o modelo alcançou uma **perda de 0,0736** e uma **acurácia de 97,26%**, indicando que ele consegue generalizar bem para novos dados. O **Gráfico 1** ilustra a evolução da perda e precisão ao longo das épocas (*epochs* no gráfico) de treinamento e validação.

Gráfico 1 - Perda e acurácia nas etapas de treino e validação do modelo.



Fonte: O autor (2024).

O modelo treinado foi avaliado em um conjunto de testes composto por imagens distintas dos conjuntos de treinamento e validação. No conjunto de teste, o modelo apresentou uma perda de 0.2020 e uma acurácia de 91,89%. Esses resultados mostram que, apesar da complexidade e variabilidade dos dados de teste, o modelo *EfficientNetV2* customizado manteve um desempenho consistente e capaz de generalizar para novos dados.

Esses resultados estão alinhados com estudos recentes que apontam a eficácia de arquiteturas modernas para tarefas de classificação de pragas, especialmente em cenários de aplicação móvel. Nesse contexto, o trabalho de Akhtar et al. (2025) sobre arquiteturas de *deep learning* otimizadas para a classificação de insetos agrícolas em dispositivos de borda (*edge*) reforça a relevância de tais abordagens. Esse não apenas selecionou modelos de alto desempenho, como o EfficientNet, mas também investigou técnicas de quantização para otimizar a implementação em plataformas móveis, destacando a capacidade de manter uma acurácia de classificação de 77,8% em dados de teste, ao mesmo tempo em que o tamanho do modelo foi reduzido de 33 MB para apenas 9,6 MB pós-treinamento, permitindo que o modelo funcione dentro de um aplicativo de celular para classificar insetos em tempo real, diretamente no aparelho e sem precisar de internet.

Essa abordagem é especialmente relevante para o projeto, que também se trata do desenvolvimento de um aplicativo com o modelo integrado. Os resultados desse estudo suportam, portanto, a escolha pela arquitetura *EfficientNetV2* para o desenvolvimento de um

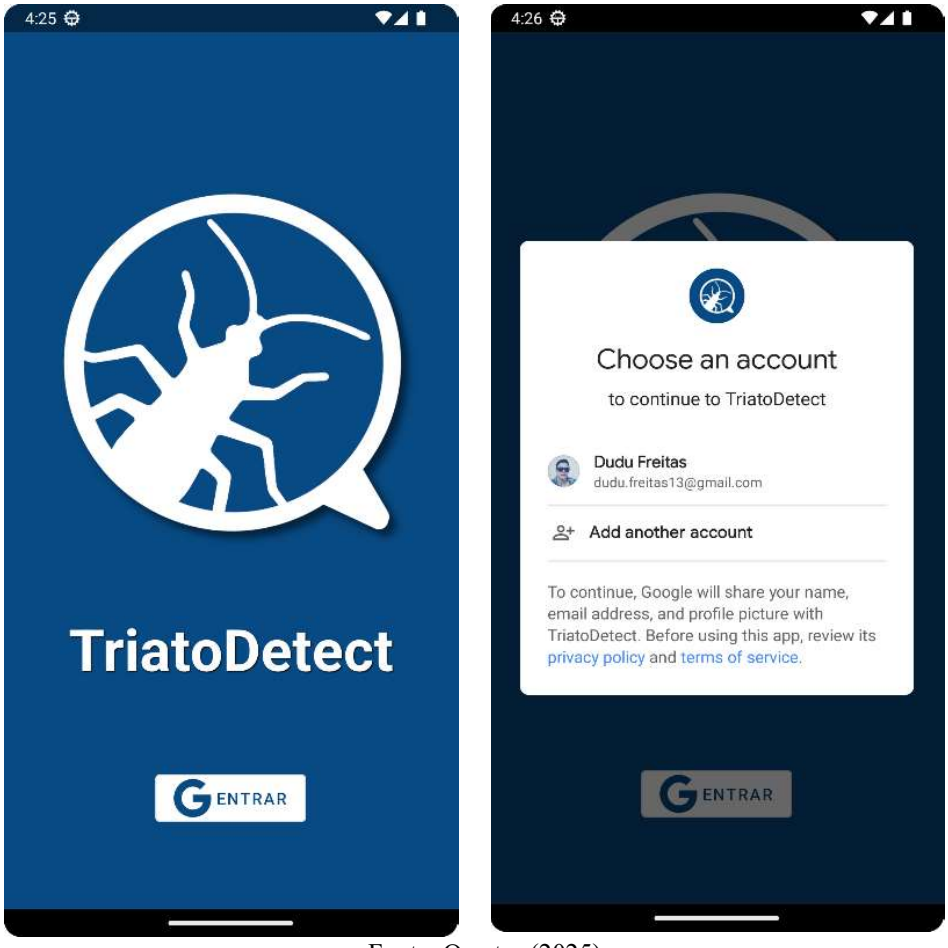
aplicativo de detecção de insetos preciso, eficiente em recursos computacionais, e que funcione bem em campo.

4.2 Aplicativo TriatoDetect

Com o modelo de classificação de imagens treinado e avaliado, a próxima etapa envolveu a integração desse modelo em o aplicativo TriatoDetect. Esse aplicativo não apenas aproveita a capacidade do modelo de identificar possíveis vetores da doença de Chagas com alta precisão, como também oferece uma interface prática e acessível ao usuário. Esta seção descreve as principais telas e funcionalidades do aplicativo.

Na **Figura 11** estão as telas de login e autenticação do aplicativo móvel TriatoDetect. A autenticação via *Firebase Authentication* usando uma conta Google disponível no aparelho em que o aplicativo foi instalado. Esta forma de autenticação foi escolhida pois os aparelhos Android necessitam de uma conta Google vinculada ao dispositivo, facilitando assim, o processo de autenticação do usuário.

Figura 11 - Ilustração da tela inicial de login (UC01) do aplicativo móvel TriatoDetect.



Fonte: O autor (2025).

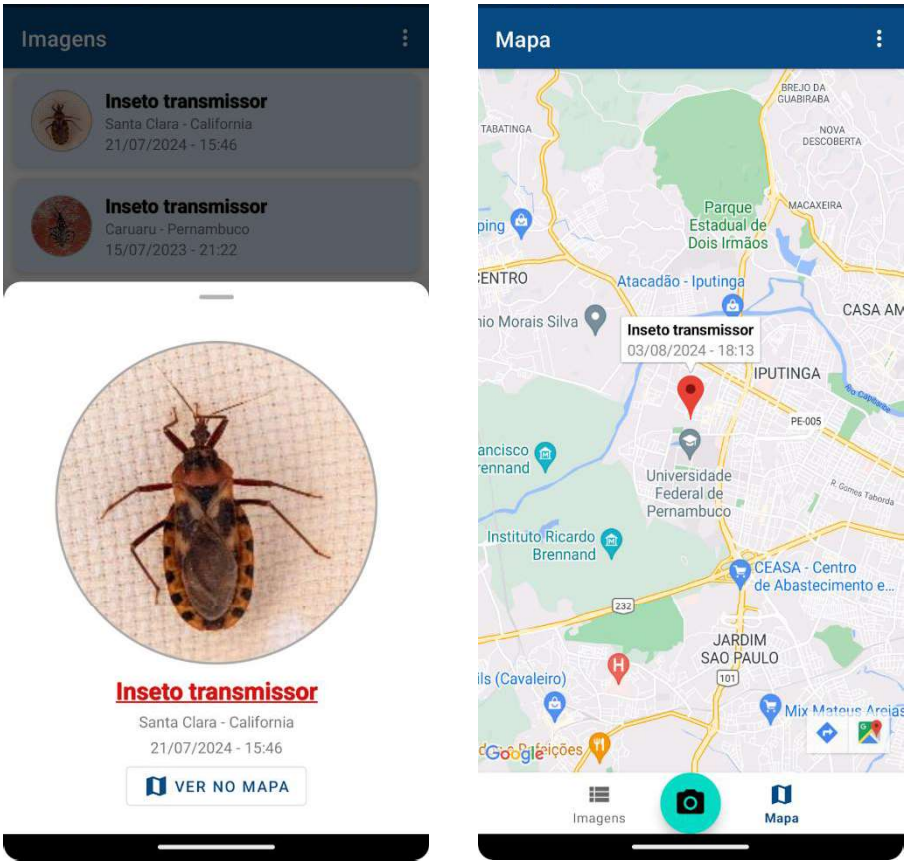
Após realizar o login, exibe-se a tela de entrada no aplicativo (**Figura 12**). Essa tela, que corresponde ao caso de uso (UC02 - Listar Imagens), possui a listagem de todas as classificações de imagem realizadas pelo usuário, além de outras funcionalidades. Ao clicar em alguma imagem da lista, é exibido um *pop-up* com todas as informações da imagem, como localização, data e hora, e a classificação.

Figura 12 - Ilustração da entrada do aplicativo (UC02) do aplicativo móvel TriatoDetect.

Fonte: O autor (2024).

No *pop-up* de informações de uma imagem (UC03 - Consultar Informações das Imagens), também possui uma ação “VER NO MAPA” (UC05 - Ver no Mapa), onde mostra no mapa a localização de onde foi feita a classificação da imagem (**Figura 13**).

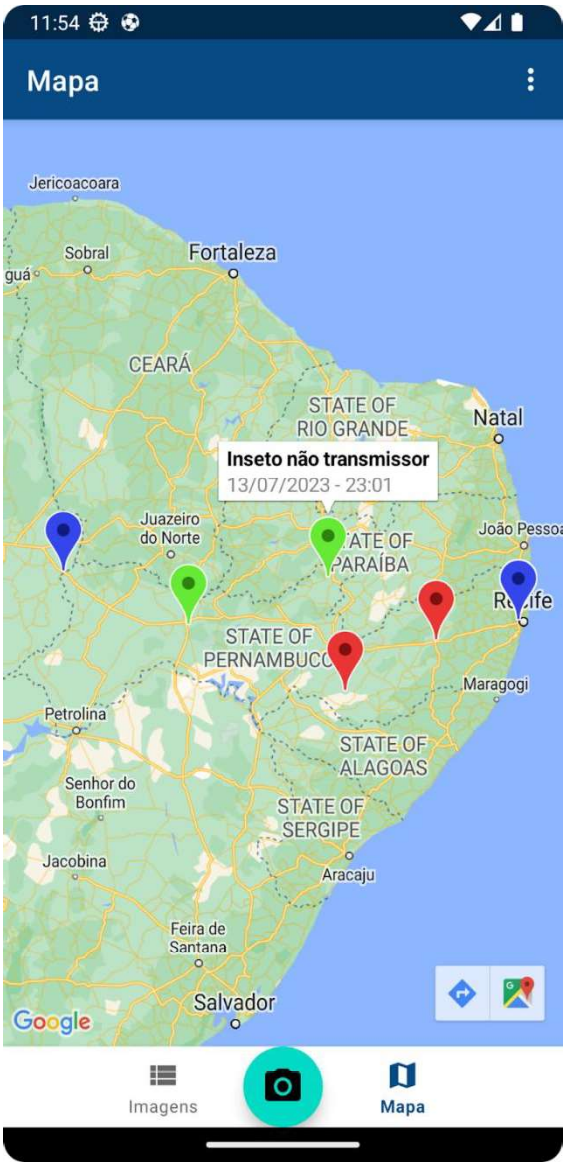
Figura 13 - Ilustração das opções dos casos de uso UC03 e UC05 do aplicativo móvel TriatoDetect.



Fonte: O autor (2024).

Voltando para a tela de listagem, existe a opção de o usuário visualizar as localizações de todas as classificações feitas pelo aplicativo, tanto as classificações relacionadas às imagens adicionadas pelo usuário, como também por outros usuários, fornecendo assim um mapa de risco, no caso de uso UC04 - Visualizar Mapa, como exemplifica na **Figura 14**.

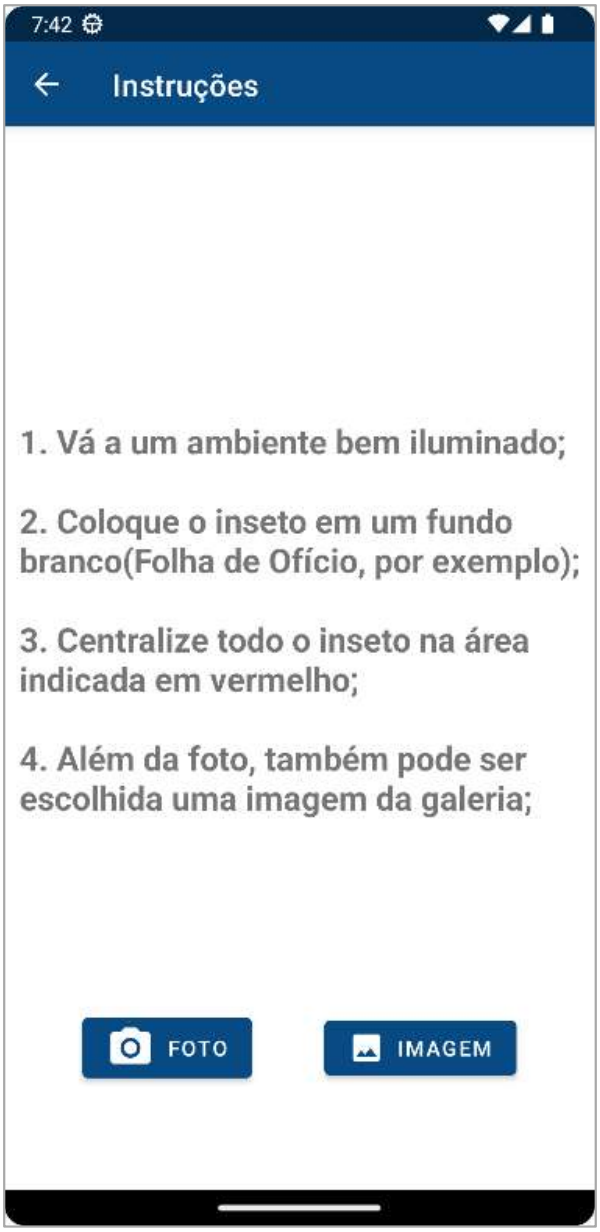
Figura 14 - Ilustração detalhada da funcionalidade do caso de uso UC05 do aplicativo móvel TriatoDetect.



Fonte: O autor (2024).

A ação para fazer a classificação de uma imagem é representada pelo ícone de uma câmera e está centralizada na parte inferior da tela. Esta ação leva a uma tela de instruções, nesta tela, além das instruções apresentadas, o usuário pode escolher se vai realizar a captura, consequentemente ativa o sistema da câmera (UC06 - Tirar Foto) ou escolher uma imagem da galeria (UC07 - Selecionar Imagem), ou tirar uma foto (**Figura 15**).

Figura 15 - Ilustração da tela de instruções que apresenta as opções do caso de uso UC06/UC07 TriatoDetect.



Fonte: O autor (2024).

Na ação de tirar uma foto, caso de uso UC06 - Tirar Foto, o usuário acessa uma tela de câmera integrada no aplicativo onde existe a indicação em vermelho, de uma área onde o inseto deve ficar (**Figura 16**). Esta tela foi feita utilizando o CameraX, uma biblioteca do Jetpack[13] criada para facilitar o desenvolvimento de apps de câmera.

[13] developer.android.com/jetpack

Figura 16 - Ilustração exibindo a funcionalidade do caso de uso UC06 aplicativo móvel TriatoDetect.

Fonte: O autor (2025).

Ao tirar a foto, ou após selecionar uma imagem da galeria, o usuário é direcionado para a tela de confirmação, onde nela, ele pode confirmar se a imagem deve ser classificada ou cancelada, voltando para a tela de instruções (**Figura 17**).

Figura 17 - Ilustração exibindo a tela de confirmação do aplicativo móvel TriatoDetect.

Fonte: O autor (2024).

Ao confirmar a imagem no caso de uso UC08 - Classificar Imagem, o processo de classificação é realizado utilizando o modelo criado neste projeto, que é executado diretamente no dispositivo do usuário utilizando uma biblioteca do TensorFlow[3]. Todo o código da classificação feita diretamente no aplicativo, está disponível no Apêndice B.

Quando a classificação estiver pronta, todos os dados são salvos no Firebase (UC09 - Enviar Imagem) e o usuário seja redirecionado para a tela de listagem, onde a nova imagem já está disponível para consulta. Ademais, o usuário pode identificar insetos classificados como transmissores da doença de Chagas através de uma coloração específica para cada classificação, detalhada no **Quadro 2**.

Quadro 2 - Classificação por cor utilizada após análise das imagens no aplicativo móvel TriatoDetect.

Classificações	**Cores**
Inseto Transmissor	Vermelho
Inseto Não Transmissor	Verde
Não identificado	Azul

Fonte: O autor (2024).

Se a classificação for de algum inseto transmissor de interesse, um e-mail é enviado para os e-mails pré-cadastrados (UC10 – Notificar Autoridades), informando que foi detectado um inseto transmissor da doença de Chagas, qual a localização e horário, além da imagem. Referente a localização presente no e-mail, esta é fornecida através de um link onde a autoridade que recebe o e-mail possa acessar, sendo redirecionada para o Google Maps com um indicador da localização exata de onde a detecção foi feita (**Figura 18**).

Figura 18 - Ilustração exibindo a tela de e-mail após detecção de um inseto transmissor através do aplicativo móvel TriatoDetect.

Fonte: O autor (2025).

4.3 Discussão

A classificação de imagens realizada pelo aplicativo (UC08), que categoriza as imagens em "Inseto transmissor", "Inseto não transmissor" ou "Não identificado", foi projetada dessa forma para facilitar o entendimento do público em geral. A intenção foi garantir que o usuário não precisasse de conhecimento prévio sobre triatomíneos para compreender a funcionalidade e a classificação fornecidas pelo aplicativo. Dessa forma, a estruturação do aplicativo foi concebida para ser acessível à população, alinhando-se com Dias et al. (2016), que destacam a importância da participação comunitária na notificação de triatomíneos para a manutenção do controle de vetores domiciliares. No entanto, tem-se observado que, com o êxito na redução da transmissão da infecção chagásica, somado ao surgimento de outros agravos com maior visibilidade na mídia e na comunicação em saúde, uma parte da população, especialmente os mais jovens, que não vivenciaram os períodos de transmissão ativa da doença de Chagas associada à alta infestação por triatomíneos, apresentam dificuldades para reconhecer e encaminhar corretamente os insetos aos serviços de saúde (Dias et al., 2016).

Além disso, o trabalho de Honorato (2020), que avaliou a presença de triatomíneos em diferentes mesorregiões do Rio Grande do Norte, destacou que a falta de identificação do ambiente e do ecótopo de captura dos triatomíneos, impossibilitaram o entendimento mais completo do panorama da epidemiologia dos vetores da doença de Chagas na região, dificultando assim a realização das atividades de vigilância e controle desses insetos (Honorato, 2020). Dessa forma, durante a elaboração dos casos de uso do aplicativo, o UC10 (Notificar Autoridades) foi incluído devido a necessidade de, além da identificação dos insetos transmissores da doença de Chagas, por vezes desconhecidos pela população, também comunicar aos setores e/ou equipes da vigilância epidemiológica e entomológica.

Ademais, Honorato (2020) também afirma que houve a entrega de insetos erroneamente pela população, que insetos hemípteros predadores e fitófagos, foram confundidos com triatomíneos. Assim sendo, ressalta-se a importância da elaboração desse aplicativo, principalmente na diferenciação dos insetos transmissores daqueles que não são transmissores da doença de Chagas, visto que há diversas semelhanças morfológicas, podendo causar dúvidas aos profissionais e à população.

A ferramenta *web* e móvel “TriatoKey”, desenvolvida por Oliveira et al. (2017), foi criada para auxiliar na identificação dos triatomíneos, voltada principalmente para técnicos de saúde. A abordagem de desenvolvimento desse software baseia-se em séries de perguntas de

"sim" ou "não" conduzindo o usuário na direção da identificação correta da espécie ou táxon. Basicamente, a identificação é estruturada em caracteres morfológicos visualizados por fotos de espécimes catalogados da Coleção de Vetores da Doença de Chagas (Fiocruz-COLVEC), bem como, vivos de colônias de insetários da Fiocruz.

Comparando-se a ferramenta desenvolvida por Oliveira et al. (2017), o "TriatoKey", com o aplicativo desenvolvido no presente trabalho, o TriatoDetect, o principal ponto de divergência são os métodos que foram utilizados para realizar a identificação dos insetos. Enquanto o "TriatoKey" se utiliza de perguntas e respostas para conduzir a identificação, o TriatoDetect utiliza um modelo de *Deep Learning* para identificação automática dos insetos, podendo este portanto, por não demandar um conhecimento prévio e detalhado da entomologia, ser explorado tanto por profissionais e quanto pela população.

Outra funcionalidade implementada no aplicativo TriatoDetect, desenvolvido no presente trabalho, é a exibição de um mapa de risco com os insetos transmissores, os Triatomíneos, detectados por todos os usuários, permitindo assim que o usuário consiga visualizar todos os locais aos quais foram identificados a presença desse vetor, consequentemente alertando às pessoas que estão nessas regiões para adoção de medidas de proteção e fomentar a vigilância. Tal funcionalidade assemelha-se ao mapa interativo elaborado pela Secretaria de Saúde do Estado de São Paulo, que exibe áreas de risco de transmissão da Febre maculosa, diante da presença dos vetores na região (São Paulo, 2024).

Por fim, pretende-se que o aplicativo desenvolvido possa auxiliar na identificação e notificação aos órgãos de vigilância responsáveis, e ajude, portanto, a superar principalmente as barreiras físicas, visto que a extensão territorial do estado de Pernambuco torna-se um obstáculo às ações de vigilância. Ademais, também é desejado que esse aplicativo contribua para o empoderamento e educação da população acerca dos insetos transmissores da doença de Chagas e possa ser mais uma ferramenta para auxiliar na identificação e captura desses vetores, podendo ele ser implementado junto aos Postos de Informação em Triatomíneos (PIT), que estão aos poucos sendo estabelecidos no Estado. Hoje há cerca de 25 PIT's, ao quais grande parte encontra-se concentrados na região da 4º Gerência Regional de Saúde, com sede em Caruaru (IV GERES) (Fiocruz, 2017).

4.4 Limitações

Apesar dos bons resultados alcançados por este trabalho e é fundamental reconhecer as limitações referentes ao estudo, que abrem caminhos para pesquisas futuras. Primeiramente, a principal limitação está no tamanho e na composição do conjunto de dados utilizado para o treinamento do modelo. Embora técnicas de aumento de dados fora utilizada para expandi-la, a base original de 450 imagens é relativamente pequena para os padrões de modelos de *deep learning*. A literatura demonstra que a performance e a capacidade de generalização de modelos de visão computacional estão diretamente correlacionadas ao volume e à diversidade dos dados de treinamento (Sun et al., 2017). A queda de acurácia observada entre o conjunto de validação (97,26%) e o de teste (91,89%) sugere um leve sobre ajuste (*overfitting*) e um desafio de generalização, onde o modelo pode não oferecer a mesma eficácia em imagens com características distintas das vistas no treino.

Em segundo lugar, o conjunto de dados, embora diversificado por imagens de ciência cidadã e com isso, possuírem uma boa variação de fotos tiradas por usuários, ainda assim, podem conter vieses de seleção que afetam o desempenho em cenários do mundo real. As imagens foram coletadas de fontes acadêmicas e principalmente, da plataforma BioDiversity4All, a quais, podem não representar completamente a distribuição de espécies e as condições de captura fotográfica (tipos de câmera de celular, iluminação, fundos) encontradas pelos usuários finais em Pernambuco. Este fenômeno, conhecido como *dataset shift*, ocorre quando a distribuição dos dados de produção difere da distribuição dos dados de treinamento, sendo um desafio conhecido na implementação de sistemas de aprendizado de máquina (Torralba & Efros, 2011; Mehrabi et al., 2021).

A terceira limitação é a ausência de uma validação de campo. O modelo foi avaliado em um conjunto de teste estático, mas sua performance ainda não verificada em um estudo clínico ou de campo, com o aplicativo sendo utilizado em tempo real por agentes de saúde e pela população. A validação em cenários reais é um passo indispensável para garantir a segurança, eficácia e confiabilidade de qualquer ferramenta de inteligência artificial aplicada à saúde antes de sua implementação em larga escala (Topol, 2019).

Apesar dessas limitações, os resultados obtidos oferecem um base para a evolução futura do modelo e aplicativo, que devem incluir a expansão contínua do banco de dados, assim como a realização de estudos de validação em campo.

5 CONSIDERAÇÕES FINAIS

O aplicativo móvel TriatoDetect, desenvolvido neste trabalho, oferece aos cidadãos uma ferramenta tendo como objetivo de facilitar a identificação dos vetores do Chagas, se utilizando de um modelo de *Deep Learning* com acurácia superior a 90% na diferenciação dos insetos transmissores e os não transmissores. Além disso, o aplicativo também notifica as autoridades sanitárias ao enviar um e-mail quando insetos transmissores da doença de Chagas são identificados. Assim sendo, objetivando superar as barreiras físicas impostas pela vasta extensão territorial do estado de Pernambuco, espera-se que o aplicativo contribua significativamente para a melhoria das ações de vigilância em saúde.

Ademais, uma das funcionalidades do TriatoDetect é a exibição de um mapa de risco que permite aos usuários visualizar as áreas onde os triatomíneos foram detectados. Isso não apenas alerta a população sobre a presença desses vetores, mas também fomenta a adoção de medidas de proteção e vigilância. E por ser um aplicativo móvel, sua acessibilidade se amplia, permitindo que uma diversidade maior de usuários tenha acesso às informações, bastando apenas um celular com internet.

Prospectivamente, avalia-se a ampliação do potencial de identificação de vetores, aumentando o banco de dados do aplicativo para incluir uma maior variedade de espécies de triatomíneos, além das quatro atualmente reconhecidas. Também seria benéfico aprimorar a funcionalidade do mapa de risco, incluindo uma função de notificação aos usuários quando triatomíneos forem detectados nas proximidades. Por fim, a criação de uma versão do aplicativo voltada para profissionais entomologistas, com recursos avançados de identificação quanto a espécie e gênero dos triatomíneos, pode potencializar ainda mais as aplicações do TriatoDetect.

Com essas propostas, espera-se que o TriatoDetect seja adotado como uma ferramenta eficaz para a promoção da saúde pública, contribuindo para a educação e empoderamento da população no combate à doença de Chagas.

REFERÊNCIAS

AKHTAR, M. H.; EKSHEIR, I.; SHANABLEH, T. Edge-Optimized Deep Learning Architectures for Classification of Agricultural Insects with Mobile Deployment. **Information**, v. 16, n. 1, p. 19, 2025. DOI: https://doi.org/10.3390/info16050348. Acesso em: 20 jun. 2025.

BRASIL. Ministério da Saúde. Secretaria de Vigilância em Saúde. Guia de Vigilância em Saúde. Brasília, 2019. p. 465-488. Disponível em: <https://bvsms.saude.gov.br/bvs/publicacoes/guia_vigilancia_saude_3ed.pdf>. Acesso em: 19 nov. 2022.

CHAIPANHA, W. & KAEWWICHIAN, P. Smote vs. Random Undersampling for imbalanced data- car ownership demand model. Communications - Scientific Letters of the University of Zilina, 24 (3) D105-D115, 2022. DOI: https://doi.org/10.26552/com.C.2022.3.D105-D115. Acesso em: 10 jan. 2024.

CHOLLET, F. Deep Learning with Python. Manning Publications Co. 2018. Disponível em: https://file.fouladi.ir/courses/deep/books/Fran%C3%A7ois%20Chollet-Deep%20Learning%20with%20Python-Manning%20(2018)-.pdf. Acesso em: 21 fev. 2024.

DIAS, J. V. L.; QUEIROZ, D. R. M.; DIOTAIUTI, L.; PIRES, H. H. R. Conhecimentos sobre triatomíneos e sobre a doença de Chagas em localidades com diferentes níveis de infestação vetorial. Ciência & Saúde Coletiva, 21 (7). 2016. DOI: 10.1590/141381232015217.07792015. Acesso em: 28 ago. 2024.

DEVI, R. S. S.; KUMAR, V. R. V.; SIVAKUMAR, P. EfficientNetV2 Model for Plant Disease Classification and Pest Recognition. Computer Systems Science and Engineering, v. 45, n. 2, p. 2249–2263, 2023. DOI: https://doi.org/10.32604/csse.2023.032231. Acesso em: 20 jun 2025.

EL-KASSAS, W. S.; ABDULLAH, B. A.; YOUSEF, A. H.; WAHBA, A. M. Taxonomy of Cross-Platform Mobile Applications Development Approaches. Ain Shams Engineering Journal. Volume 8, 2, 2017, p. 163-190. DOI: https://doi.org/10.1016/j.asej.2015.08.004. Acesso em: 01 mar. 2024.

FIOCRUZ. Entregue pessoalmente - PITsMaps ferramenta que fornece a localização do Posto de Informação em Triatomíneos (PIT). 2017. Disponível em: <https://chagas.fiocruz.br/entregue-pessoalmente/>. Acesso em: 26 set. 2024.

GURGEL - GONÇALVES, R.; KOMP, E.; CAMPBELL, L. P.; KHALIGHIFAR, A.; MELLENBRUCH, J.; MENDONÇA, V. J.; OWENS, H. L.; FELIX, K. L. C.; PETERSON, A. T.; RAMSEY, J. M. Automated identification of insect vectors of Chagas disease in Brazil and Mexico: the Virtual Vector Lab. PeerJ 5: e3040, 2017. DOI: https://doi.org/10.7717/peerj.3040. Acesso em: 28 nov. 2022.

GOOGLE. Android Studio. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 26 fev. 2024.

HONORATO, N. R. M. Avaliação da presença de Triatomíneos e distribuição de DTUs do *Trypanosoma Cruzi* em diferentes mesorregiões do Rio Grande do Norte, Brasil. Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Biociências. Programa de Pós-Graduação em Biologia Parasitária. 2020. Disponível em: <https://repositorio.ufrn.br/bitstream/123456789/29280/1/Avaliacaopresencatriatomineos_Honorato_2020.pdf>. Acesso em: 09 set. 2024.

JANSEN, A. M.; XAVIER, S. C. C.; ROQUE, A. L. R. (2020). Landmarks of the knowledge and Trypanosoma cruzi biology in the wild environment. Frontiers in Cellular and Infection Microbiology. DOI: https://doi.org/10.3389/fcimb.2020.00010. Acesso em: 19 nov.2022.

MEHRABI, N.; MORSTATTER, F.; SAXENA, N.; LERMAN, K.; GALSTYAN, A. A Survey on Bias and Fairness in Machine Learning. ACM Computing Surveys, v. 54, n. 6, p. 1–35, 2021. DOI: https://doi.org/10.48550/arXiv.1908.09635. Acesso em: 23 jun. 2025.

MENDES, P. C.; LIMA, S. C.; PAULA, M. B. C.; SOUZA, A. A.; RODRIGUES, E. A. S.; LIMONGI, J. E. Doença de Chagas e a distribuição espacial de triatomíneos capturados em Uberlândia, Minas Gerais – Brasil. Hygeia 3(6):176 - 204, Jun/2008. Disponível em: https://seer.ufu.br/index.php/hygeia/article/view/16905/9316. Acesso em: Acesso em: 14 jul. 2023.

MEDEIROS, C. A.; SILVA, M. B. A.; OLIVEIRA, A. L. S.; ALVES, S. M. M.; JÚNIOR, W. O.; MEDEIROS, Z. M. Spatial analysis of the natural infection index for Triatomines and the risk of Chagas disease transmission in Northeastern Brazil. Rev. Inst. Med. trop. S. Paulo,65, 2023. DOI: https://doi.org/10.1590/S1678-9946202365032. Acesso em: 27 jul. 2023.

MENEZES, K. R. Guia ilustrado de triatomíneos do estado de Pernambuco [dissertação]. São Paulo: Universidade de São Paulo, Faculdade de Saúde Pública; 2018. Disponível em: <https://www.teses.usp.br/teses/disponiveis/6/6142/tde-12042019101417/publico/KellyReisdeMenezesREVISADA.pdf>. Acesso em: 15 maio 2024.

NAGAHAMA, A. Learning and predicting the unknown class using evidential deep learning. *Sci Rep* 13, 14904, 2023. DOI: https://doi.org/10.1038/s41598-023-40649-w. Acesso em: 28 maio 2024.

OLIVEIRA, L. M.; BRITO, R. N.; GUIMARÃES, P. A. S.; SANTOS, R. V. M. A.; DIOTAIUTI, L. G.; SOUZA, R. C. M.; RUIZ, J. C. TriatoKey: a web and mobile tool for biodiversity identification of Brazilian triatomine species. DATABASE - The Journal of Biological Databases and Curation, Volume 2017. DOI: https://doi.org/10.1093/database/bax033. Acesso em: 24 set. 2024.

PARSONS, Z.; BANITAAN, S.; AL-REFAI, G.; Automated Chagas Disease Vectors Identification using Data Mining Techniques. Conference: 2020 IEEE International Conference on Electro Information Technology (EIT)At: Chicago, IL, USA. DOI: 10.1109/EIT48999.2020.9208261. Acesso em: 20 out. 2024.

SÃO PAULO. Secretaria de Saúde do Estado de São Paulo. Febre Maculosa Brasileira. Disponível em: <https://capital.sp.gov.br/web/saude/w/vigilancia_em_saude/doencas_e_agravos/240753>. Acesso em: 24 set. 2024.

SILVA, L. R. S.; SILVA, M. B. A.; OLIVEIRA, G. M. A.; MEDEIROS, C. A.; OLIVEIRA, J. B. Vigilância entomológica dos vetores da doença de Chagas nos municípios da VIII Gerência Regional de Saúde do estado de Pernambuco, Brasil, de 2012 a 2017. Revista Pan-Amazônica de Saúde, vol.12. Ananindeua, 2021. Disponível em: <http://scielo.iec.gov.br/scielo.php?script=sci_arttext&pid=S2176-62232021000100015>. Acesso em: 19 nov. 2022.

SILVA, M. B. A.; DE MENEZES, K. R.; SIQUEIRA, A. M.; BALBINO, V. de Q.; LOROSA, E. S.; DE FARIAS, M. C. G.; FREITAS, M. T. S.; DA SILVA, A.; PORTELA, V. M. C.; JURBERG, J. Importância da distribuição geográfica dos vetores da doença de Chagas em Pernambuco, Brasil, em 2012. Revista de Patologia Tropical / Journal of Tropical Pathology, Goiânia, v. 44, n. 2, p. 195–206, 2015. DOI: https://doi.org/10.5216/rpt.v44i2.36650. Acesso em: 17 mar. 2023.

SILVA, M. B. A.; BORBA, R. F. B.; FERREIRA, G. M. O. G.; MEDEIROS, C. A.; ROCHA, D. S. Avaliação externa da qualidade da identificação entomológica de triatomíneos realizada na Rede de Laboratórios Públicos em Pernambuco, 2017. Epidemiologia e Serviços de Saúde, 30(2), 2021. Disponível em: <https://www.scielo.br/j/ress/a/k65jYvwxhBGhX97wx8wk3hk/?lang=pt>. Acesso em: 19 nov. 2022.

SILVA, M. B. A.; ROCHA, D. S.; BORBA, R. F. B. Triatomíneos Sinantrópicos de Pernambuco: Biogeografia, Técnicas laboratoriais e controle da qualidade [recurso eletrônico]. UPE-EDUPE, p. 30-37, Recife, 2019. Disponível em: <https://www.edupe.upe.br/images/livros/Triatomineos_Sinantropicos_de_PE.pdf>. Acesso: 10 mar. 2023.

SILVA, L. R. S.; SILVA, M. B. A.; OLIVEIRA, G. M. A.; MEDEIROS, C. A.; OLIVEIRA, J. B. Vigilância entomológica dos vetores da doença de Chagas nos municípios da VIII Gerência Regional de Saúde do estado de Pernambuco, Brasil, de 2012 a 2017. Rev Pan-Amaz Saude vol.12, Ananindeua, 2021. DOI: http://dx.doi.org/10.5123/s2176-6223202100858. Acesso em: 10 mar. 2023.

SILVA, T.R.M. Distribuição espacial e caracterização molecular das linhagens de *Trypanosoma cruzi* em triatomíneos em municípios do Agreste de Pernambuco. Tese (Doutorado) - Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Biociência Animal, Recife, 2022. Disponível em: <http://www.tede2.ufrpe.br:8080/tede/bitstream/tede2/8646/2/Tatiene%20Rossana%20Mota%20Silva.pdf>. Acesso em: 18 mar. 2023.

SOUZA, I. C. A.; RODRIGUES, F. C. S.; ARAÚJO, A. P.; SOUZA, J. M. B. S.; DIOTAIUTI, L. G.; FERREIRA, R. A. Moradores de áreas rurais de municípios mineiros endêmicos para a doença de Chagas: ideias e concepções sobre a doença, os vetores e os serviços de saúde. Cad Saúde Colet, 2023; 31 (3):e31030595. DOI: https://doi.org/10.1590/1414-462X202331030595. Acesso em: 23 nov. 2022.

STATCOUNTER. StatCounter: GlobalStats. Mobile Operating System Market Share Brazil, Setembro 2023 - Setembro 2024. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/brazil>. Acesso em: 13 fev. 2024.

SUN, C.; SHRIVASTAVA, A.; SINGH, S.; GUPTA, A. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017. DOI: https://doi.org/10.1109/ICCV.2017.97. Acesso em: 23 jun. 2025.

TOPOL, E. J. High-performance medicine: the convergence of human and artificial intelligence. Nature Medicine, v. 25, n. 1, p. 44–56, 2019. DOI: https://doi.org/10.1038/s41591-018-0300-7. Acesso em: 23 jun. 2025.

TORRALBA, A.; EFROS, A. A. Unbiased look at dataset bias. In: CVPR 2011, p. 1521-1528. IEEE, 2011. Disponível em: https://people.csail.mit.edu/torralba/publications/datasets_cvpr11.pdf. Acesso em: 23 jun. 2025.

WORLD HEALTH ORGANIZATION. Control of Chagas disease. WHO Technical Report Series 905, 109. 2018. Disponível em: <https://www.ricet.es/media/30090/lancet5.pdf>. Acesso em: 19 nov. 2022.

APÊNDICE A

```
# Instala as bibliotecas necessárias: keras-tuner para otimização de hiperparâmetros e roboflow para manipulação do dataset
!pip install keras-tuner --q
!pip install roboflow --q
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 3.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.7/86.7 kB 5.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.8/66.8 kB 6.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.9/49.9 MB 19.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 121.5 MB/s eta 0:00:00
```

```
# Importa as bibliotecas essenciais para o desenvolvimento do modelo de aprendizado de máquina
import numpy as np
import keras
from keras import layers
from tensorflow import data as tf_data
import matplotlib.pyplot as plt
from keras import applications
import tensorflow as tf
import numpy as np
import os
import keras_tuner as kt
from roboflow import Roboflow
import json
```

```
# Exibe informações sobre a GPU disponível no ambiente de execução
!nvidia-smi
```

```
Sat Jun 21 00:54:11 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   49C    P8             10W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

```
# Define constantes e parâmetros para o modelo, como tamanho da imagem, tamanho do lote, número de classes e rótulos
IMG_SIZE = 224
IMAGE_SIZE = (IMG_SIZE, IMG_SIZE)
BATCH_SIZE = 32
CLASSES = 3
LABELS_CLS = ["n", "s", "u"] # n-> não transmisso / s -> transmissor / u -> não definido
INITIAL_EPOCHS = 25 # Número inicial de épocas para a busca de hiperparâmetros
FINAL_EPOCHS = 0 # Número de épocas para o fine-tuning do melhor modelo
```

```
# Obtém o diretório de trabalho atual e cria um diretório chamado 'datasets' para armazenar os dados
import os
HOME = os.getcwd()
print(HOME)

!mkdir {HOME}/datasets
# Altera o diretório atual para a pasta 'datasets'
%cd {HOME}/datasets
```

```
/content
/content/datasets
```

```
# Baixa o dataset do Roboflow utilizando a chave de API e detalhes do projeto fornecidos
rf = Roboflow(api_key="vuaR7tYv7NeXYuonalwQ")
project = rf.workspace("ifpetcc").project("class-uni")
version = project.version(1)
dataset = version.download("folder")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in CLASS-UNI-1 to folder:: 100%|██████████| 28520/28520 [00:00<00:00, 41056.75it/s]

Extracting Dataset Version Zip to CLASS-UNI-1 in folder:: 100%|██████████| 378/378 [00:00<00:00, 2544.17it/s]
```

```
# Define os caminhos para as pastas dos conjuntos de dados de treino, validação e teste
train_folder_dataset = "/content/datasets/CLASS-UNI-1/train"
val_folder_dataset= "/content/datasets/CLASS-UNI-1/valid"
test_folder_dataset= "/content/datasets/CLASS-UNI-1/test"


# Carrega o conjunto de dados de treino a partir do diretório especificado
train_ds = keras.utils.image_dataset_from_directory(
      train_folder_dataset,
      validation_split=None, # Não divide para validação aqui, pois temos um conjunto de validação separado
      subset=None, # Usa o diretório inteiro como conjunto de treino
      seed=1337, # Semente para embaralhar e garantir reprodutibilidade
      label_mode="int", # Os rótulos são codificados como int
      image_size=IMAGE_SIZE, # Redimensiona as imagens para o tamanho definido
      batch_size=BATCH_SIZE # Define o tamanho do lote
  )
```

```
Found 254 files belonging to 3 classes.
```

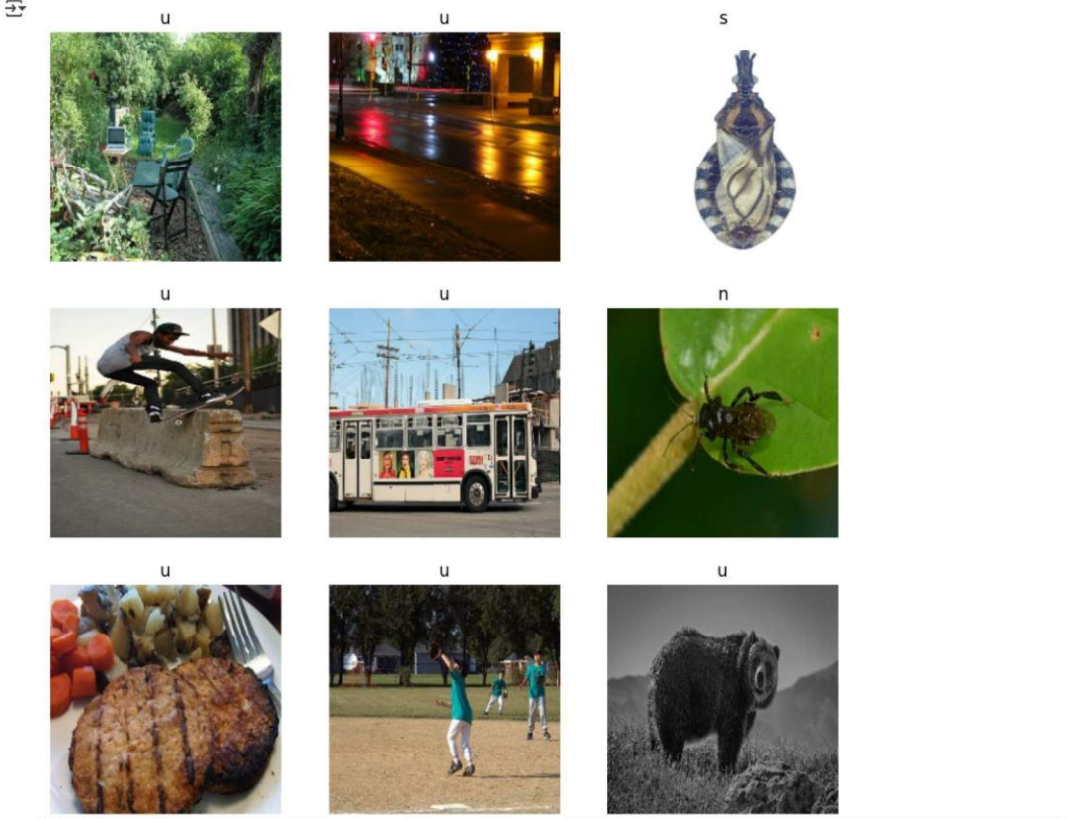
```
# Carrega o conjunto de dados de validação a partir do diretório especificado
val_ds = keras.utils.image_dataset_from_directory(
      val_folder_dataset,
      validation_split=None, # Não é necessário dividir para validação
      subset=None, # Usa o diretório inteiro como conjunto de validação
      seed=1337, # Semente para reprodutibilidade
      label_mode="int", # Os rótulos são codificados como int
      image_size=IMAGE_SIZE, # Redimensiona as imagens
      batch_size=BATCH_SIZE # Define o tamanho do lote
  )
```

```
Found 73 files belonging to 3 classes.
```

```
# Carrega o conjunto de dados de teste a partir do diretório especificado
test_ds = keras.utils.image_dataset_from_directory(
      test_folder_dataset,
      validation_split=None, # Não é necessário dividir para validação
      subset=None, # Usa o diretório inteiro como conjunto de teste
      seed=1337, # Semente para reprodutibilidade
      label_mode="int", # Os rótulos são codificados como int
      image_size=IMAGE_SIZE, # Redimensiona as imagens
      batch_size=BATCH_SIZE # Define o tamanho do lote
  )
```

```
Found 37 files belonging to 3 classes.
```

```
# Exibe algumas imagens de exemplo do conjunto de dados de treino
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(np.array(images[i]).astype("uint8"))
        plt.title(LABELS_CLS[int((labels[i]))]) # Exibe o rótulo da classe
        plt.axis("off") # Oculta os eixos
```




```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal_and_vertical"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
        layers.RandomTranslation(0.1, 0.1),
        layers.RandomContrast(0.1),
        layers.RandomBrightness(0.1),
    ],
    name="data_augmentation",
)


train_ds = train_ds.prefetch(tf_data.AUTOTUNE)
val_ds = val_ds.prefetch(tf_data.AUTOTUNE)
test_ds = test_ds.prefetch(tf_data.AUTOTUNE)


print("Especificação do Dataset de Treino:", train_ds.element_spec)
print("Especificação do Dataset de Validação:", val_ds.element_spec)
```

```
Especificação do Dataset de Treino: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,),
Especificação do Dataset de Validação: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,
```

```
# Constrói o modelo base
def build_base_model(input_shape=(224, 224, 3)):
    # --- Pipeline do Modelo Base---
    # 1. Camada de Input
    inputs = layers.Input(shape=input_shape)

    # 2. Aumento de Dados (executado na GPU, apenas no treinamento)
    x = data_augmentation(inputs)

    # 3. Pré-processamento específico do modelo base
    x = applications.efficientnet_v2.preprocess_input(x)

    # 4. Modelo Base Congelado
```

```
    base_model = applications.EfficientNetV2B0(
        include_top=False,
        weights='imagenet',
        input_tensor=x,
    )

    return base_model, inputs

base_model, inputs = build_base_model()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b0_notop.h5
24274472/24274472 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
def build_model(hp, input_shape=(224, 224, 3), num_classes=3):
    # --- Pipeline do Modelo ---
    base_model.trainable = False

    # 1. Cabeça de Classificação Leve (Melhor Prática)
    x = layers.GlobalAveragePooling2D(name="avg_pool")(base_model.output)
    x = layers.BatchNormalization()(x) # Ajuda a estabilizar o treinamento

    hp_dropout_rate = hp.Float('dropout_rate', min_value=0.2, max_value=0.5, step=0.1)
    x = layers.Dropout(hp_dropout_rate, name="dropout")(x)

    # Camada de saída
    outputs = layers.Dense(num_classes, activation='softmax', name="predictions")(x)

    # --- Fim do Pipeline ---

    # Criação e Compilação
    model = keras.Model(inputs, outputs)

    hp_learning_rate = hp.Float("lr", min_value=1e-5, max_value=1e-2, sampling="log")
    optimizer = keras.optimizers.Adam(learning_rate=hp_learning_rate)

    # Compilando o Model
    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")],
    )
    return model


# Inicializa o tuner Hyperband para a otimização de hiperparâmetros
tuner = kt.Hyperband(
    build_model, # A função que constrói o modelo
    objective=[kt.Objective('val_loss', 'min'), kt.Objective('val_acc', 'max')], # Objetivos a serem otimizados (minimizar perda de val
    max_epochs=INITIAL_EPOCHS, # Número máximo de épocas para treinar
    factor=3, # Fator pelo qual o número de épocas é reduzido em brackets sucessivos
    directory='my_dir', # Diretório para salvar os resultados
    project_name='tcc_triato' # Nome do projeto
)


# Define um callback de EarlyStopping para parar o treino se a perda de validação não melhorar
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

# Inicia a busca pelos melhores hiperparâmetros utilizando o tuner
tuner.search(train_ds, epochs=INITIAL_EPOCHS, validation_data=val_ds, callbacks=[stop_early])

# Obtém os melhores hiperparâmetros encontrados pelo tuner
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]


# Obtém os melhores hiperparâmetros como um dicionário e os salva em um arquivo JSON
best_hps_dict = best_hps.values
with open('best_hyperparameters.json', 'w') as json_file:
    json.dump(best_hps_dict, json_file)


# Define o caminho para o arquivo JSON com os melhores hiperparâmetros salvos no Google Drive
file_path = '/content/drive/MyDrive/triat/best_hyperparameters_efficientnet_v2.json'

# Carrega os melhores hiperparâmetros do arquivo JSON
with open(file_path, 'r') as json_file:
    best_hps_dict = json.load(json_file)

# Converte o dicionário de volta para um objeto HyperParameters do KerasTuner
best_hps = kt.HyperParameters()
best_hps.values = best_hps_dict
```

```
# Exibe os melhores hiperparâmetros carregados
best_hps.values
```

```
{'dropout_rate': 0.30000000000000004,
 'lr': 0.0024646295953947225,
 'tuner/epochs': 25,
 'tuner/initial_epoch': 0,
 'tuner/bracket': 0,
 'tuner/round': 0}
```

```
# Constrói o modelo utilizando os melhores hiperparâmetros encontrados durante a otimização
model = tuner.hypermodel.build(best_hps)

# Define callbacks para o treinamento: ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss', # Monitora a perda de validação
    factor=0.1, # Reduz a taxa de aprendizado por um fator de 0.1
    patience=3, # Número de épocas sem melhora após o qual a taxa de aprendizado será reduzida
    min_lr=1e-6 # Limite inferior para a taxa de aprendizado
)

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='best_model.keras', # Caminho para salvar o melhor modelo
    monitor='val_loss', # Monitora a perda de validação
    save_best_only=True, # Salva apenas o melhor modelo
    mode='min', # Salva quando a perda de validação for mínima
    verbose=1 # Exibe mensagens ao salvar
)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',      # Monitora a perda de validação
    patience=7,              # Número de épocas a esperar sem melhora antes de parar o treino.
    verbose=1,               # Imprime uma mensagem na tela quando o treino é interrompido.
    restore_best_weights=True # Restaura os pesos do modelo da melhor época encontrada.
)

# Treina o modelo utilizando os melhores hiperparâmetros e os callbacks definidos
history = model.fit(
    train_ds,
    epochs=INITIAL_EPOCHS, # Treina pelo número inicial de épocas
    callbacks=[checkpoint_callback, reduce_lr, early_stopping_callback], # Usa os callbacks definidos
    validation_data=val_ds, # Usa o conjunto de validação para avaliação
)
```

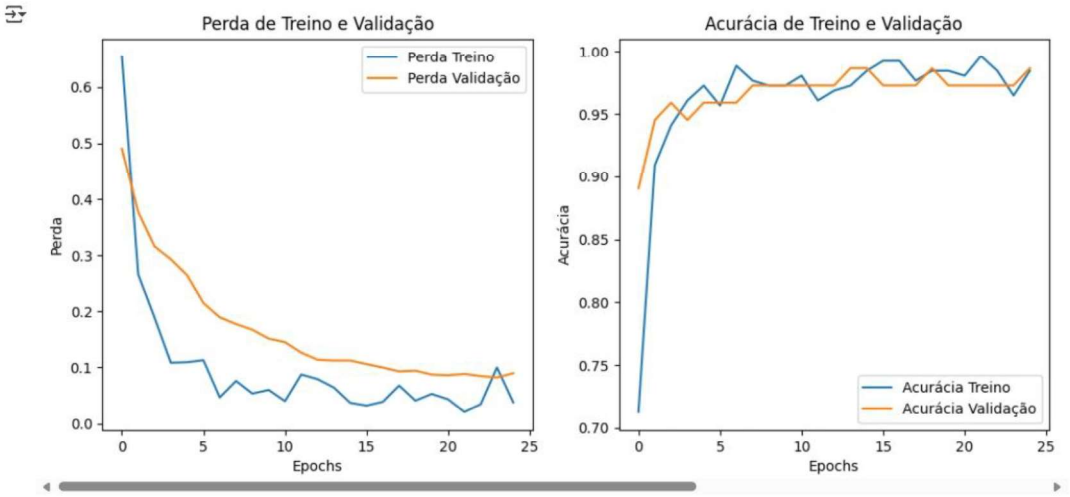
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - acc: 0.9953 - loss: 0.0245
Epoch 22: val_loss did not improve from 0.08602
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 134ms/step - acc: 0.9936 - loss: 0.0241 - val_acc: 0.9726 - val_loss: 0.0884 - learning_rate: 0.0025
Epoch 23/25
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - acc: 0.9871 - loss: 0.0239
Epoch 23: val_loss improved from 0.08602 to 0.08460, saving model to best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 240ms/step - acc: 0.9868 - loss: 0.0250 - val_acc: 0.9726 - val_loss: 0.0846 - learning_rate: 0.0025
Epoch 24/25
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - acc: 0.9886 - loss: 0.0358
Epoch 24: val_loss improved from 0.08460 to 0.08204, saving model to best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 239ms/step - acc: 0.9859 - loss: 0.0429 - val_acc: 0.9726 - val_loss: 0.0820 - learning_rate: 0.0025
Epoch 25/25
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - acc: 0.9898 - loss: 0.0216
Epoch 25: val_loss did not improve from 0.08204
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - acc: 0.9892 - loss: 0.0234 - val_acc: 0.9863 - val_loss: 0.0896 - learning_rate: 0.0025
Restoring model weights from the end of the best epoch: 24.
```

```
# Carrega o melhor modelo salvo durante o treinamento
def load_best_model(best_model_path='best_model.keras'):
    # Define objetos customizados necessários para carregar o modelo
    custom_objects = {'preprocess_input': applications.efficientnet_v2.preprocess_input}

    # Carrega o modelo
    best_model = tf.keras.models.load_model(
        best_model_path,
        custom_objects=custom_objects,
        safe_mode=False # Define como False para permitir o carregamento de objetos customizados
    )

    return best_model

model = load_best_model()
```

```
# Imprime o último valor de cada métrica do histórico de treinamento
history_dict = history.history

for key, values in history_dict.items():
    print(f"{key}: {values[-1]}")  # Exibindo o último valor de cada métrica
```

```
acc: 0.9842519760131836
loss: 0.03755103796720505
val_acc: 0.9863013625144958
val_loss: 0.08964268118143082
learning_rate: 0.0024646297097206116
```

```
# Plota os gráficos de perda e acurácia de treino e validação
import matplotlib.pyplot as plt

# Extrai os dados de perda e acurácia do histórico
loss = history_dict['loss']
val_loss = history_dict['val_loss']
accuracy = history_dict.get('acc')
val_accuracy = history_dict.get('val_acc')

# Cria o gráfico de perda
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(loss, label='Perda Treino')
plt.plot(val_loss, label='Perda Validação')
plt.title('Perda de Treino e Validação')
plt.xlabel('Epochs')
plt.ylabel('Perda')
plt.legend()

# Cria o gráfico de acurácia, se disponível
if accuracy and val_accuracy:
    plt.subplot(1, 2, 2)
    plt.plot(accuracy, label='Acurácia Treino')
    plt.plot(val_accuracy, label='Acurácia Validação')
    plt.title('Acurácia de Treino e Validação')
    plt.xlabel('Epochs')
    plt.ylabel('Acurácia')
    plt.legend()

plt.show()
```




```
# Método para exibir as camadas do modelo
def show_layers(model_show):
    for i, layer in enumerate(model_show.layers):
      print(f"{i:<5} {layer.name:<25} {str(layer.trainable):<20}")

# Metodo para descongelar as ultimas camadas de um modelo
def unfreeze_base_model(prefixo_inicial: str):
    base_model.trainable = False
    print("Modelo base inicialmente congelado.")

    show_layers(base_model)

    encontrou_ponto_de_corte = False
    camadas_descongeladas = 0

    print(f"\n--- Procurando ponto de corte com prefixo: '{prefixo_inicial}' ---")

    for layer in base_model.layers:
      if encontrou_ponto_de_corte:
        layer.trainable = True
        print(f"  - Descongelando (pós-corte): {layer.name}")
        camadas_descongeladas += 1
      elif layer.name.startswith(prefixo_inicial):
        encontrou_ponto_de_corte = True
        layer.trainable = True
        print(f"  - PONTO DE CORTE ENCONTRADO! Descongelando: {layer.name}")
        camadas_descongeladas += 1

    if not encontrou_ponto_de_corte:
      print(f"\nAVISO: O prefixo '{prefixo_inicial}' não foi encontrado em nenhuma camada.")
    else:
      print(f"\nOperação concluída. Total de {camadas_descongeladas} camadas descongeladas.")

    show_layers(base_model)


# Descongelando ultimas camadas do modelo
unfreeze_base_model("block5")

# Compila o modelo para fine-tuning com uma taxa de aprendizado menor e função de perda SparseCategoricalCrossentropy
model.compile(
    optimizer=keras.optimizers.Adam(1e-5), # Usa o otimizador Adam com uma taxa de aprendizado menor
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=False), # Usa entropia cruzada esparsa categórica como função de perda
    metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")] # Monitora a acurácia categórica esparsa (corrigido para 3 classes)
)

# Define o callback ReduceLROnPlateau para fine-tuning
reduce_lr_ft = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss', # Monitora a perda de validação
    factor=0.2, # Reduz a taxa de aprendizado por um fator de 0.2
    patience=3, # Número de épocas sem melhora
    min_lr=1e-8 # Limite inferior para a taxa de aprendizado
)

# Define o callback ModelCheckpoint para fine-tuning
checkpoint_callback_ft = tf.keras.callbacks.ModelCheckpoint(
```

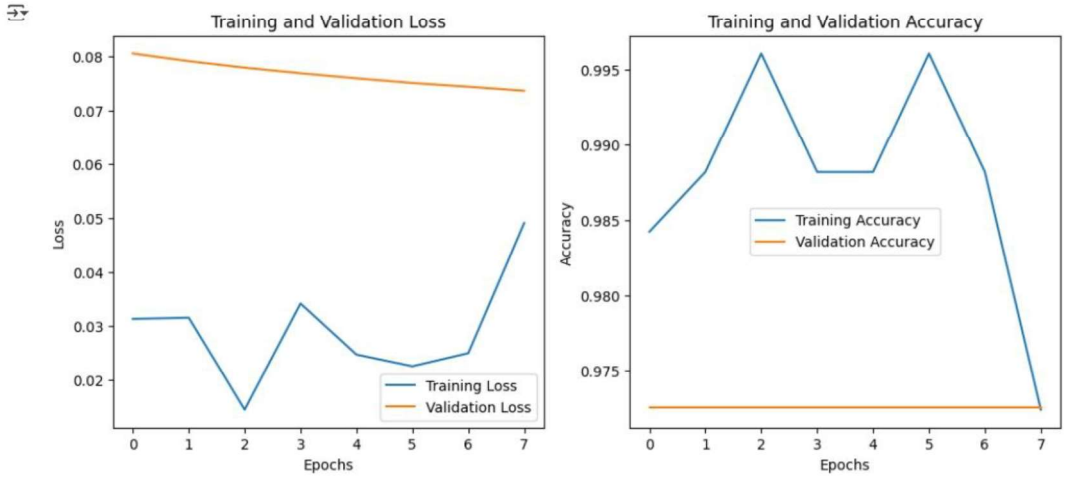
```
    filepath='fine_tuned_best_model.keras', # Caminho para salvar o melhor modelo
    monitor='val_loss', # Monitora a perda de validação
    save_best_only=True, # Salva apenas o melhor modelo
    mode='min', # Salva quando a perda de validação for mínima
    verbose=1 # Exibe mensagens ao salvar
)

# Define o callback EarlyStopping para fine-tuning
early_stopping_ft = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=8, # Paciência maior para dar chance ao ReduceLROnPlateau
    verbose=1,
    restore_best_weights=True # Essencial!
)

# Realiza o fine-tuning do modelo
history_fine = model.fit(train_ds, validation_data=val_ds,
                         epochs=history.epoch[-1] + FINAL_EPOCHS, # Treina por épocas adicionais
                         initial_epoch=history.epoch[-1], # Inicia a partir da última época do treinamento anterior
                         callbacks=[reduce_lr_ft, checkpoint_callback_ft, early_stopping_ft] # Usa os callbacks definidos
                         )
```

```
246   block6g_se_reduce         True
247   block6g_se_expand         True
248   block6g_se_excite         True
249   block6g_project_conv      True
250   block6g_project_bn        True
251   block6g_drop              True
252   block6g_add               True
253   block6h_expand_conv       True
254   block6h_expand_bn         True
255   block6h_expand_activation True
256   block6h_dwconv2           True
257   block6h_bn                True
258   block6h_activation        True
259   block6h_se_squeeze        True
260   block6h_se_reshape        True
261   block6h_se_reduce         True
262   block6h_se_expand         True
263   block6h_se_excite         True
264   block6h_project_conv      True
265   block6h_project_bn        True
266   block6h_drop              True
267   block6h_add               True
268   top_conv                  True
269   top_bn                    True
270   top_activation            True
Epoch 25/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - acc: 0.9896 - loss: 0.0214
Epoch 25: val_loss improved from inf to 0.08047, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 19s 681ms/step - acc: 0.9890 - loss: 0.0225 - val_acc: 0.9726 - val_loss: 0.0805 - learning_rate: 1.0000
Epoch 26/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - acc: 0.9753 - loss: 0.0542
Epoch 26: val_loss improved from 0.08047 to 0.07907, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 326ms/step - acc: 0.9767 - loss: 0.0516 - val_acc: 0.9726 - val_loss: 0.0791 - learning_rate: 1.0000e
Epoch 27/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step - acc: 0.9965 - loss: 0.0088
Epoch 27: val_loss improved from 0.07907 to 0.07787, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 356ms/step - acc: 0.9965 - loss: 0.0094 - val_acc: 0.9726 - val_loss: 0.0779 - learning_rate: 1.0000e
Epoch 28/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - acc: 0.9799 - loss: 0.0590
Epoch 28: val_loss improved from 0.07787 to 0.07683, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 243ms/step - acc: 0.9808 - loss: 0.0562 - val_acc: 0.9726 - val_loss: 0.0768 - learning_rate: 1.0000e
Epoch 29/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - acc: 0.9920 - loss: 0.0211
Epoch 29: val_loss improved from 0.07683 to 0.07591, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 250ms/step - acc: 0.9916 - loss: 0.0215 - val_acc: 0.9726 - val_loss: 0.0759 - learning_rate: 1.0000e
Epoch 30/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - acc: 0.9933 - loss: 0.0238
Epoch 30: val_loss improved from 0.07591 to 0.07505, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 243ms/step - acc: 0.9936 - loss: 0.0236 - val_acc: 0.9726 - val_loss: 0.0751 - learning_rate: 1.0000e
Epoch 31/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - acc: 0.9923 - loss: 0.0226
Epoch 31: val_loss improved from 0.07505 to 0.07436, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 241ms/step - acc: 0.9918 - loss: 0.0228 - val_acc: 0.9726 - val_loss: 0.0744 - learning_rate: 1.0000e
Epoch 32/32
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step - acc: 0.9714 - loss: 0.0477
Epoch 32: val_loss improved from 0.07436 to 0.07362, saving model to fine_tuned_best_model.keras
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 372ms/step - acc: 0.9715 - loss: 0.0478 - val_acc: 0.9726 - val_loss: 0.0736 - learning_rate: 1.0000e
Restoring model weights from the end of the best epoch: 32.
```

```
# Imprime o último valor de cada métrica do histórico de fine-tuning
history_fine_dict = history_fine.history

for key, values in history_fine_dict.items():
    print(f"{key}: {values[-1]}")  # Exibindo o último valor de cada métrica
```

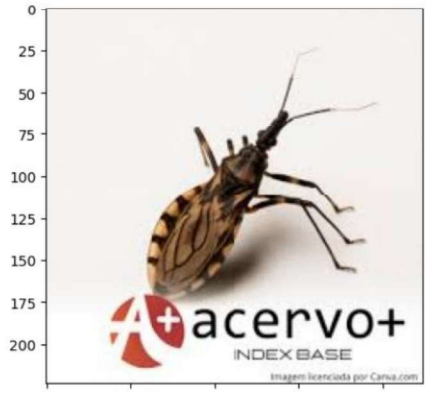
```
acc: 0.9724409580230713
loss: 0.04905371740460396
val_acc: 0.9726027250289917
val_loss: 0.07361606508493423
learning_rate: 9.999999747378752e-06
```

```
# Plota os gráficos de perda e acurácia de treino e validação para o fine-tuning
import matplotlib.pyplot as plt

# Extrai os dados de perda e acurácia do histórico de fine-tuning
loss = history_fine_dict['loss']
val_loss = history_fine_dict['val_loss']
accuracy = history_fine_dict.get('acc')
val_accuracy = history_fine_dict.get('val_acc')

# Cria o gráfico de perda
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Cria o gráfico de acurácia, se disponível
if accuracy and val_accuracy:
    plt.subplot(1, 2, 2)
    plt.plot(accuracy, label='Training Accuracy')
    plt.plot(val_accuracy, label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

plt.show()
```




```
# Carrega melhor após o fine tunning
best_model = load_best_model('fine_tuned_best_model.keras')


# Carrega uma imagem, a pré-processa e faz uma predição utilizando o modelo treinado
img = keras.utils.load_img("tb_in.jpg", target_size=IMAGE_SIZE) # Carrega a imagem e a redimensiona
plt.imshow(img) # Exibe a imagem

img_array = tf.keras.preprocessing.image.img_to_array(img) # Converte a imagem para um array NumPy
img_array = tf.expand_dims(img_array, 0)  # Adiciona uma dimensão de lote

predictions = best_model.predict(img_array) # Realiza a predição
index = np.argmax(predictions) # Obtém o índice da classe predita
print("Class:", LABELS_CLS[index]) # Imprime o rótulo da classe predita
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
Class: s
```




```
# Avalia o desempenho do modelo no conjunto de dados de teste
results = best_model.evaluate(test_ds)
print(f"Loss: {results[0]}") # Imprime a perda no conjunto de teste
print(f"Accuracy: {results[1]}") # Imprime a acurácia no conjunto de teste
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 552ms/step - acc: 0.9251 - loss: 0.1894
Loss: 0.20204779505729675
Accuracy: 0.9189189076423645
```

APÊNDICE B

```
//Dependência do Tensorflow para classificação
implementation 'org.tensorflow:tensorflow-lite:+'
implementation 'org.tensorflow:tensorflow-lite-support:+'
```

```
// Constantes/Variaveis Importante
private const val pathModel: String = "model/model_detection_triatominies_float32.tflite"
private const val THRESHOLD: Float = 0.7f
private const val IMAGE_RESIZE: Int = 224
var result: MutableList<String> = ArrayList()
```

```
fun initClassify(context: Context, bytes: ByteArray, user: User?, callback: (Boolean) -> Unit) {
    // Executa a classificação em uma thread de background
    GlobalScope.launch(Dispatchers.IO) {
        try {
            // Limpando resultados antigos
            result.clear()

            // Convertendo a imagem em bytes para Bitmap
            val bitmap: Bitmap = BitmapFactory.decodeByteArray(bytes, offset: 0, bytes.size)

            // Classificação executada em background thread
            classifyMultiClass(context, bitmap)

            // Redimensiona e comprime a imagem antes de salvar
            val compressedImageBytes = resizeAndCompressImage(bitmap)

            // Volta para a main thread para executar operações de UI e Firebase
            withContext(Dispatchers.Main) {
                saveImageStores(compressedImageBytes, user, context) { result ->
                    callback(result)
                }
            }

        } catch (e: Exception) {
            e.printStackTrace()
            // Volta para a main thread para executar o callback
            withContext(Dispatchers.Main) {
                callback(false)
            }
        }
    }
}
```

```kotlin
// Classifica uma imagem bitmap em múltiplas classes usando o modelo TFLite
private fun classifyMultiClass(context: Context, bitmap: Bitmap) {
    // Carrega o modelo previamente mapeado em memória
    val model = loadModelFile(context)
    // Pré-processa a imagem para transformar no formato aceito pelo modelo (normalização, redimensionamento etc.)
    val input = preProcessImageClassify(bitmap)
    // Cria o interpretador do TensorFlow Lite com o modelo carregado
    val interpreter = Interpreter(model)
    // Cria um array de saída para armazenar as previsões (1 linha, 3 classes)
    val output = Array( size: 1) { FloatArray( size: 3) }
    // Executa a inferência do modelo com o input e preenche o array de saída
    interpreter.run(input, output)
    // Obtém o array de previsões da primeira (e única) amostra
    val prediction = output[0]
    // Encontra o valor máximo da previsão, que representa a classe mais provável
    val maxValue = prediction.maxOrNull() ?: throw IllegalArgumentException("Array is empty")
    // Se o valor máximo for menor que o limiar definido, considera que não há previsão confiável
    if(maxValue < THRESHOLD) {
        result.add("u")      // 'u' indicando "Não identificado" ou "unknown"
        result.add("1.0")    // valor padrão
        return
    }
    // Determina a classe correspondente ao valor máximo
    when (prediction.toList().indexOf(maxValue)) {
        0 -> result.add("n")  // Inseto não Transmissor
        1 -> result.add("s")  // Inseto Transmissor
        else -> {             // caso caia fora das classes conhecidas
            result.add("u")
            result.add("1.0")
        }
    }
    // Adiciona o valor da confiança (probabilidade) ao resultado
    result.add(maxValue.toString())
}
```

```kotlin
// Carrega o modelo TFLite a partir dos assets e mapeia em memória para uso pelo TensorFlow Lite
@Throws(Exception::class)
private fun loadModelFile(context: Context): MappedByteBuffer {
    // Abre o arquivo do modelo dentro da pasta "assets"
    val fileDescriptor = context.assets.openFd(pathModel)

    // Cria um FileInputStream a partir do descritor do arquivo
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)

    // Obtém o canal do arquivo, necessário para mapear o arquivo em memória
    val fileChannel = inputStream.channel

    // Obtém o offset inicial do arquivo (posição onde o modelo começa)
    val startOffset = fileDescriptor.startOffset

    // Obtém o tamanho declarado do arquivo
    val declaredLength = fileDescriptor.declaredLength

    // Mapeia o arquivo em memória (somente leitura) e retorna o MappedByteBuffer
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength)
}
```

```
private fun preProcessImageClassify(bitmap: Bitmap): Array<Array<Array<FloatArray>>> {
    // Redimensiona a imagem para o tamanho esperado pelo modelo (IMAGE_RESIZE x IMAGE_RESIZE)
    val resizedBitmap = bitmap.scale(IMAGE_RESIZE, IMAGE_RESIZE, filter: true)
    // Cria o array de entrada para o modelo:
    // Estrutura: [1 amostra][altura][largura][3 canais RGB]
    val input = Array( size: 1) { Array(IMAGE_RESIZE) { Array(IMAGE_RESIZE) { FloatArray( size: 3) } } }
    // Percorre cada pixel da imagem redimensionada
    for (i in 0 ≤ until < IMAGE_RESIZE) {
        for (j in 0 ≤ until < IMAGE_RESIZE) {
            // Obtém o valor do pixel na posição (i, j)
            val pixel = resizedBitmap[i, j]
            // Extrai os valores de vermelho, verde e azul do pixel
            // e armazena no array de entrada do modelo
            input[0][i][j][0] = Color.red(pixel).toFloat()    // canal vermelho
            input[0][i][j][1] = Color.green(pixel).toFloat()  // canal verde
            input[0][i][j][2] = Color.blue(pixel).toFloat()   // canal azul
        }
    }
    // Retorna o array tridimensional pronto para passar ao interpretador TFLite
    return input
}
```