Detecção por Imagem (YOLOv8)

Introdução

Métodos de aprendizado profundo têm sido amplamente utilizados para a tarefa em questão. As redes neurais convolucionais (CNNs) são componentes essenciais do aprendizado profundo e são amplamente utilizadas para a detecção de objetos em imagens, devido à sua capacidade de lidar com dados multidimensionais (DU et al., 2023). As CNNs consistem em sistemas nos quais “neurônios” artificiais estão interconectados, trocando informações entre si. Essas redes são compostas por várias camadas de neurônios, em que cada neurônio responde a combinações de entradas provenientes das camadas anteriores. O aprendizado dessas redes ocorre por meio do ajuste dos pesos durante o processo de treinamento (HIJAZI; KUMAR; ROWEN, 2015).. Segundo Silva, Teixeira e Frances (2023), as arboviroses transmitidas pelo mosquito *Aedes*, como Dengue, Zika e Chikungunya, são um problema de saúde pública, para contrarrestar isto, propuseram o uso de algoritmos de detecção de objetos (YOLO) com redes neurais convolucionais (CNNs) para identificar e classificar larvas do mosquito. A abordagem alcançou bons resultados, com 85,6% de mAP e erro médio quadrático (MSE) de 0,70024, indicando desempenho satisfatório. Oliveira et al. (2023) apresenta um sistema inteligente baseado em Visão Computacional e Internet das Coisas (IoT) para a detecção em tempo real do mosquito *Aedes aegypti*. Utilizando o algoritmo YOLOv7, o sistema demonstrou desempenho superior em relação a métodos tradicionais, alcançando uma acurácia de 97%. Este trabalho combinou detecção precisa com coleta contínua de dados por meio de uma arquitetura IoT. Tendo em vista a grande quantidade de trabalhos e datasets destinados à tarefa identificação de mosquitos, propõe-se o uso do modelo YOLO V8 para identificação de mosquitos do tipo *Aedes Aegypti*.

Dataset

Contar com um dataset de imagens é parte fundamental para o treinamento de qualquer algoritmo de detecção de objetos. Pela extrema relevância do tema, existem diversos datasets de imagens de mosquitos *Aedes Aegypti* em plataformas de machine learning como Kaggle ou Roboflow. Para realizar o treinamento, primeiro foi escolhida uma base de dados que concordasse com os nossos objetivos. Nesse contexto, foi escolhida o dataset contido no projeto “mosquito Computer Vision Project”, disponível na página Roboflow. Este dataset contém 3552 imagens, sendo 70% para treinamento, 20% para validação e 10% para teste. As etiquetas são duas: “aedes” e “non_aedes”, ou seja, o algoritmo detecta estas duas classes de mosquitos.

Treinamento

Para o treinamento da rede com dados personalizados, utilizou-se a plataforma Google Colab, que oferece acesso gratuito a recursos computacionais, sendo especialmente útil para tarefas de aprendizado de máquina e ciência de dados. Devido às limitações do plano básico da plataforma, o modelo foi treinado por apenas 80 épocas. O primeiro passo para realizar o treinamento é estabelecer uma conexão com a API do Roboflow, autenticando o acesso com a chave API fornecida (no caso, "a0T2wJrHefPJmOlvYn42"). Isso permite a interação com projetos, versões e datasets armazenados na plataforma Roboflow. Em seguida, acessa-se o workspace chamado "researchmosquito" dentro da conta da equipe e, posteriormente, o projeto específico chamado "mosquito-hpxxz". No Roboflow, os dados são organizados dentro de workspaces e projetos. Por fim, realiza-se o download da versão 1 do projeto no formato YOLOv8. O Roboflow oferece diferentes formatos de exportação, sendo "yolov8" um dos formatos compatíveis com o modelo YOLOv8. Isso significa que as imagens e anotações do dataset serão baixadas no formato adequado para treinar o modelo YOLOv8.

O código utilizado para essa tarefa é apresentado a seguir:

```
# -*- coding: utf-8 -*-
"""DENGUE.ipynb
Automatically generated by Colab.
Original file is located at
https://colab.research.google.com/drive/1d28jYJ5iC2TtGYJqaeFsPKp6KluhiUs8
"""

# MY_SECRET_KEY=""

!pip install ultralytics
!pip install roboflow

import ultralytics
from roboflow import Roboflow
from ultralytics import YOLO
from IPython.display import Image

rf = Roboflow(api_key="SUA_API_KEY_AQUI")
project = rf.workspace("researchmosquito").project("mosquito-hpxxz")
version = project.version(1)
dataset = version.download("yolov8")

from ultralytics import YOLO

# Carrega o modelo YOLOv8 pré-treinado (por exemplo, YOLOv8n, YOLOv8s, YOLOv8m, etc.)
model = YOLO("yolov8n.pt") # Use seu próprio modelo treinado: "runs/detect/train/weights/best.pt"

# Mostrar a arquitectura del modelo
print(model.model)

!yolo task=detect mode=train model=yolov8s.pt
data=/content/mosquito-1/data.yaml epochs=500 imgsz=640 plots=True

# VIEW MODEL TRANING CHARTS
Image(filename=f'/content/runs/detect/train/results.png', width=600)

# VALIDATION
!yolo task=detect mode=val model=/content/runs/detect/train/weights/best.pt
data=/content/mosquito-1/data.yaml

# Passo 1: Compactar a pasta /content/runs em um arquivo .zip
!zip -r /content/runs.zip /content/runs

# Passo 2: Fazer o download do .zip
from google.colab import files
files.download('/content/runs.zip')
```

Deploy

Por enquanto, o algoritmo está sendo testado no notebook pessoal, utilizando uma câmera web para captação de vídeo. Um detalhe importante do algoritmo de detecção, é que a inferência pode ser realizada tanto em vídeo quanto em imagens. O código realiza a captura de vídeo em tempo real e a detecção de objetos utilizando um modelo YOLO pré-treinado. Primeiramente, o modelo é carregado a partir de um arquivo de pesos e "aquecido" com uma previsão inicial, preparando-o para a detecção. O código configura diretórios de saída para salvar os resultados, o vídeo resultante e o arquivo de relatório. Em seguida, inicia a captura de vídeo com a webcam e define a largura, altura e FPS dos frames. Utilizando o OpenCV, um escritor de vídeo é configurado para salvar o vídeo com as detecções em formato MP4. A cada frame capturado, o modelo realiza a detecção de objetos e, quando detectados, salva as imagens com as detecções, registra informações como timestamp, número de objetos detectados e nome do arquivo no relatório. A contagem total de detecções e a taxa de FPS são exibidas na tela. Ao final do processo, o relatório é completado com informações gerais sobre o tempo de gravação, a quantidade total de detecções, o tamanho do arquivo de vídeo e a memória utilizada. O código finaliza liberando os recursos de captura e gravação, e fechando as janelas abertas. Dessa forma, o modelo YOLOv8 realiza a detecção e registra as informações em tempo real, criando um relatório e salvando as imagens e vídeos das detecções para análise posterior.

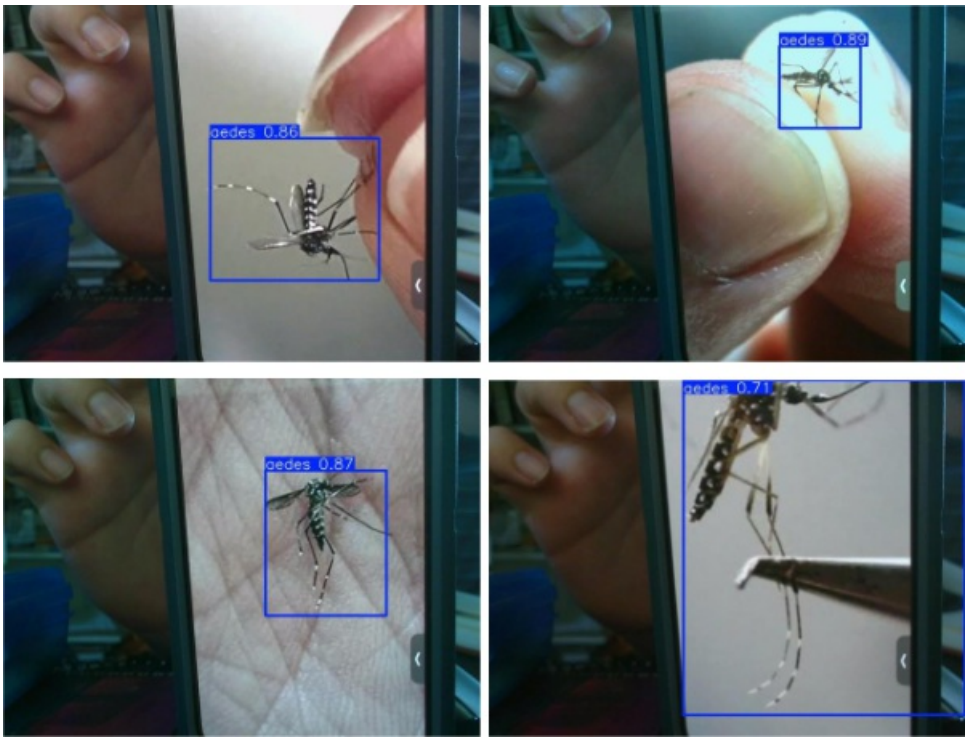
O código utilizado consta a seguir:

```
# -*- coding: utf-8 -*-
"""dengue_deploy.ipynb
Automatically generated by Colab.
Original file is located at
https://colab.research.google.com/drive/1kXsD1Ocp8-23ERrrb6TVepNgz2gzRzXY
"""
#from ultralytics import YOLO
# Cargar el modelo entrenado
#model = YOLO(r'C:\Users\lopez\OneDrive\Escritorio\DENGUE\best-80.pt')
########## Modelo com relatorio 2

from ultralytics import YOLO
import cv2
import os
import time
import numpy as np
import psutil
import os.path as path

# Carregar o modelo
modelo = YOLO(r'C:\Users\lopez\OneDrive\Escritorio\DENGUE\best-80.pt')

# Aquece o modelo
_ = modelo.predict(np.zeros((480, 640, 3), dtype=np.uint8), imgsz=480,
conf=0.70, stream=False, verbose=False)

# Diretórios
diretorio_saida =
r'C:\Users\lopez\OneDrive\Escritorio\DENGUE\resultados\prediccion2'
os.makedirs(diretorio_saida, exist_ok=True)
video_saida = os.path.join(diretorio_saida, 'video_prediccao.mp4')
```

```
relatorio_path = os.path.join(diretorio_saida, 'relatorio.txt')

# Iniciar a captura de vídeo
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

largura_frame = int(cap.get(3))
altura_frame = int(cap.get(4))
fps = int(cap.get(cv2.CAP_PROP_FPS)) or 24

# Criar o escritor de vídeo
out = cv2.VideoWriter(
    video_saida,
    cv2.VideoWriter_fourcc(*'mp4v'),
    fps,
    (largura_frame, altura_frame)
)

frame_id = 0
total_deteccoes = 0
start_time = time.time() # Início da gravação

# Abrir o arquivo de relatório
with open(relatorio_path, 'w') as relatorio:
    relatorio.write('RELATÓRIO DE DETECÇÕES\n')
    relatorio.write('========================\n\n')

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

    frame_start = time.time()

    resultados = modelo.predict(
        source=frame,
        imgsz=480,
        conf=0.70,
        stream=False,
        verbose=False
    )

    for r in resultados:
        frame_end = time.time()
        fps_atual = 1 / (frame_end - frame_start + 1e-6) # Evitar divisão por zero

        if r.boxes and len(r.boxes) > 0:
            total_deteccoes += 1
            timestamp = time.strftime("%Y%m%d-%H%M%S")
            nome_arquivo = f'deteccao_{timestamp}_{frame_id}.jpg'
            caminho_arquivo = os.path.join(diretorio_saida, nome_arquivo)
            cv2.imwrite(caminho_arquivo, r.plot()) # Salvar imagem

            # Escrever no relatório
            relatorio.write(f'Deteção {total_deteccoes}:\n')
            relatorio.write(f' - Timestamp: {timestamp}\n')
            relatorio.write(f' - Arquivo: {nome_arquivo}\n')
            relatorio.write(f' - Objetos detectados: {len(r.boxes)}\n\n')
            relatorio.flush() # Salvar imediatamente

            frame_id += 1

        texto = f'Deteções: {total_deteccoes} | FPS: {fps_atual:.1f}'
        cv2.putText(
            r.plot(), texto, (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2
        )

        out.write(r.plot())
        cv2.imshow('Predição', r.plot())

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Tempo total de gravação
    tempo_gravacao = time.time() - start_time

    # Estatísticas gerais
    peso_video = path.getsize(video_saida) / (1024 * 1024) # Em MB
    memoria_usada = psutil.virtual_memory().used / (1024 * 1024) # Em MB

    # Escrever estatísticas no relatório
    relatorio.write('\n\n')
    relatorio.write('========================\n')
    relatorio.write(f'Tempo total de gravação: {tempo_gravacao / 60:.2f} minutos\n')
    relatorio.write(f'Quantidade total de detecções: {total_deteccoes}\n')
    relatorio.write(f'Peso do arquivo de vídeo: {peso_video:.2f} MB\n')
    relatorio.write(f'Memória utilizada (aproximadamente): {memoria_usada:.2f} MB\n')

# Finalizar
cap.release()
out.release()
cv2.destroyAllWindows()
```

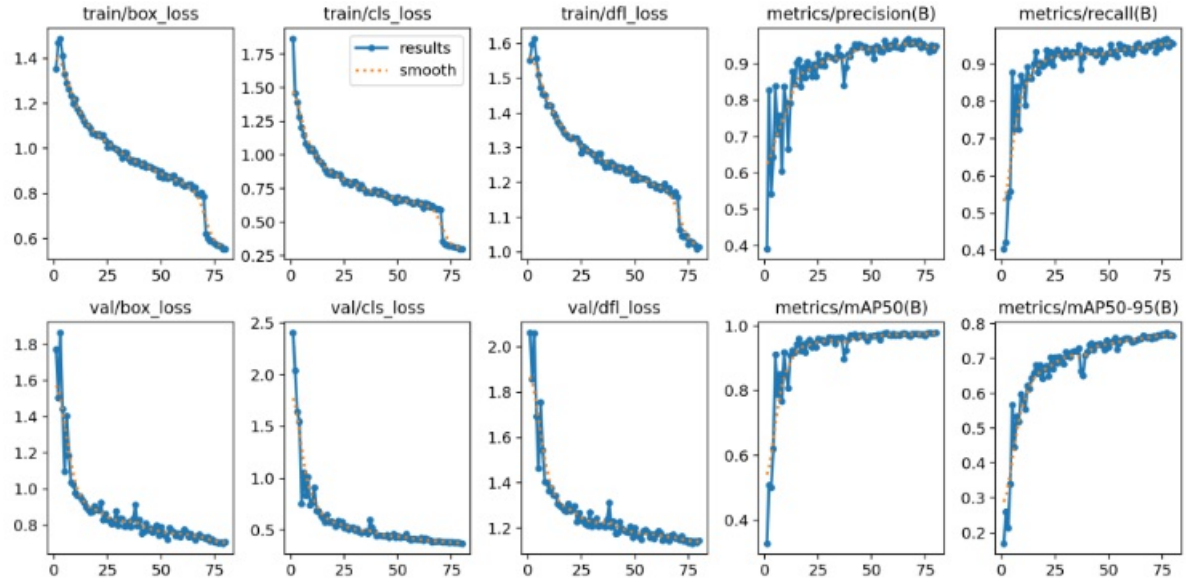
Resultados

O modelo foi testado por meio de vídeos de mosquitos gravados em um celular. Para isso, o celular foi posicionado na frente da câmera web, permitindo a realização da inferência em tempo real com o vídeo sendo capturado diretamente. Durante o teste, o modelo YOLO foi capaz de realizar a detecção de objetos (mosquitos) nos frames do vídeo, e os resultados da detecção foram registrados em imagens e vídeos, como parte do processo de avaliação. A seguir, os resultados das detecções realizadas pelo modelo são apresentados.



Métricas de Avaliação

Durante o treinamento do modelo, gráficos foram gerados para ilustrar as variações de métricas importantes, como box loss, classification loss, distribution focal loss, precisão (precision), recall e mean average precision (mAP), tanto para os conjuntos de treinamento quanto de validação ao longo das épocas de treinamento. Esses gráficos fornecem uma visão detalhada da performance do modelo, destacando como ele evolui e ajusta seus parâmetros para melhorar a detecção e a classificação dos objetos ao longo do tempo. As métricas de perda (loss) indicam a eficiência do modelo em minimizar os erros durante o processo de treinamento, enquanto as métricas de precisão, recall e mAP fornecem uma avaliação da qualidade das predições do modelo em termos de sua capacidade de detectar corretamente os objetos e minimizar falsos positivos e negativos.



Referências bibliográficas

- DU, Xing et al. A comparative study of different CNN models and transfer learning effect for underwater object classification in side-scan sonar images. *Remote Sensing*, v. 15, n. 3, p. 593, jan. 2023. DOI: https://doi.org/10.3390/rs15030593.
- HIJAZI, Samer; KUMAR, Rishi; ROWEN, Chris. Using convolutional neural networks for image recognition. 2015.
- SILVA, Romário; TEIXEIRA, Carlos de Mattos; FRANCES, Carlos. Modelo de classificação e detecção de larvas de mosquitos *Aedes* usando YOLO. [S.l.], 29 out. 2023.
- OLIVEIRA, Danilo et al. Application of YOLOv7 for real-time detection of *Aedes aegypti*. [S.l.: s.n.], 11 out. 2023.
- ROBOFLOW. *Mosquito Detection Dataset – mosquito-hpxxz*. Roboflow Universe, 2023. Disponível em:

https://universe.roboflow.com/researchmosquito/mosquito-hpxxz. Acesso em: 25 abril 2025.

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	01/05/2025	Alejandro Lopez

Identificação por Som (Batimento de Asas)

Introdução

Durante o desenvolvimento deste projeto, foi proposto ao grupo o desafio de criar uma armadilha inteligente para mosquitos, capaz não apenas de capturá-los, mas também de trazer informações relevantes para o controle vetorial. Com isso em mente, surgiu a oportunidade de integrar ao sistema um algoritmo previamente desenvolvido individualmente por um dos integrantes do grupo, que tem como função a identificação automática de espécies de mosquitos com base no som do batimento de suas asas.

Esse algoritmo se apresenta como uma solução inovadora, leve e eficiente para reconhecer espécies vetoras de doenças com base em sua assinatura acústica. Ele foi originalmente criado em outro projeto de pesquisa, mas devido à sua grande correlação com o propósito da armadilha inteligente, tornou-se uma ferramenta ideal para compor o sistema proposto neste trabalho.

Origem da Ideia e Justificativa de Uso

A ideia de utilizar esse algoritmo no projeto surgiu da convergência entre dois trabalhos distintos: por um lado, o desafio proposto pela banca, que envolvia a construção de uma armadilha automatizada; e por outro, o algoritmo de identificação acústica, desenvolvido para outro contexto, mas que se mostrou perfeitamente compatível com o novo escopo.

A integração dos dois projetos oferece um ganho significativo em valor social e técnico, pois além de capturar os mosquitos, a armadilha passará a ser capaz de identificá-los e contabilizá-los por espécie, possibilitando análises populacionais locais em tempo real. Essa funcionalidade é de grande importância para órgãos de saúde pública que precisam monitorar áreas de risco e agir de forma estratégica no combate às arboviroses.

Como o Algoritmo Será Usado no Projeto

O algoritmo será embarcado em um microcomputador Raspberry Pi, que estará acoplado à armadilha dodecaédrica. Esse sistema embarcado ficará responsável por:

- Captar o som do mosquito após sua entrada na armadilha;
- Processar esse som para gerar uma assinatura acústica única (fingerprint);
- Comparar a assinatura com uma base de dados interna previamente construída;
- Identificar automaticamente a espécie do mosquito capturado;
- Armazenar ou transmitir essa informação para um sistema de registro e monitoramento.

Todo o processamento será feito localmente, garantindo baixo consumo energético e independência de rede externa.

Como o Algoritmo Funciona

O funcionamento do algoritmo se baseia na extração de padrões acústicos contidos nos sons do batimento de asas dos mosquitos. Os passos principais incluem:

1. Captação e pré-processamento do som: o sinal é reamostrado, filtrado e convertido em um espectrograma;
2. Extração de picos espectrais: os picos mais relevantes em termos de frequência e tempo são identificados;
3. Criação do fingerprint: pares de picos são organizados em uma matriz que representa a “impressão digital” do som;
4. Matching com a base de dados: essa impressão digital é comparada com fingerprints já armazenados, extraídos de amostras conhecidas de Aedes aegypti, Aedes albopictus e Anopheles arabiensis;
5. Identificação da espécie: a espécie com o maior número de correspondências é retornada como resultado final.

Esse processo permite realizar uma identificação rápida, leve e sem necessidade de redes neurais complexas, o que é ideal para dispositivos de campo de baixo custo.

Riscos Relacionados ao Algoritmo

Apesar da robustez do algoritmo, alguns riscos operacionais precisam ser considerados:

- Ruídos externos: sons do ambiente (vento, voz humana, motores) podem interferir na captação do sinal do mosquito;

- Precisão limitada: o algoritmo pode não atingir uma taxa de acerto suficientemente alta em todas as condições ambientais, especialmente na diferenciação entre espécies com padrões sonoros semelhantes;
- Dependência da qualidade do microfone: se o microfone embarcado não tiver sensibilidade adequada, o fingerprint pode ser comprometido.

Medidas Mitigadoras

Como forma de mitigar esses riscos, o projeto prevê o desenvolvimento paralelo de um algoritmo complementar de identificação por imagem, que poderá ser utilizado como sistema de apoio ou verificação cruzada em casos de incerteza. Isso aumenta a confiabilidade geral da armadilha inteligente e abre espaço para fusão de dados em futuras versões do sistema.

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	01/05/2025	Luis Felipe Rivera

Backlog

Introdução

Este documento apresenta o **backlog específico da parte algorítmica** do projeto **Armadilha Inteligente para Aedes aegypti**. No qual o foco está no desenvolvimento do sistema de **classificação automática do mosquito**, baseado em sinais de áudio e imagem captados em tempo real.

Tabela backlog

ID	Épico	História de Usuário	Prioridade
E01	Identificação de Mosquitos	Como usuário, quero que o algoritmo classifique o mosquito pela frequência de batimento das asas.	Alta
E02.1	Captura e Processamento de Áudio	Como usuário, quero capturar o som de mosquitos com um microfone para que o algoritmo possa analisá-lo.	Alta
E02.2	Captura e Processamento de Áudio	Como usuário, quero que o algoritmo extraia a frequência dos sons capturados para identificação precisa.	Alta
E03.1	Análise de Frequência	Como usuário, quero que o algoritmo compare a frequência extraída com a faixa típica do Aedes aegypti.	Alta
E03.2	Análise de Frequência	Como usuário, quero que sons que não correspondem a batidas de asas sejam descartados automaticamente.	Média
E04	Gestão de Base de Dados	Como usuário, quero que o algoritmo carregue uma base de dados de frequências conhecidas para melhorar a acurácia.	Média
E05	Visualização de Dados	Como usuário, quero gerar um espectrograma a partir do som captado para facilitar a análise.	Média

Critérios de aceitação

1. **O algoritmo deve classificar o mosquito através da frequência do batimento das asas**

1.1. O algoritmo deve retornar um resultado categórico indicando se o som pertence ou não ao Aedes aegypti.
1.2. O algoritmo deve ter taxa de acerto mínima de 90% em testes com amostras reais previamente rotuladas.
1.3. O algoritmo deve distinguir entre diferentes espécies de mosquito com base em diferenças de frequência.
1.4. O algoritmo deve incluir uma margem de tolerância ao ruído.

2. **O algoritmo deve receber como entrada o sinal de áudio captado por um microfone**

2.1. O algoritmo deve aceitar entrada de microfone em tempo real e/ou arquivos de áudio.
2.2. O sistema deve ser capaz de processar áudio captado em tempo real ou carregado de um arquivo.

3. **O algoritmo deve extrair a frequência do som das asas do mosquito a partir do sinal de áudio**

3.1. A extração de frequência deve identificar a frequência fundamental do sinal associado ao batimento de asas.
3.2. A técnica de extração deve funcionar corretamente mesmo em presença de ruídos leves.
3.3. A extração deve ser concluída em menos de 1 segundo.

4. **O algoritmo deve comparar a frequência extraída com a faixa de frequência característica do mosquito da dengue**

(Aedes aegypti)

4.1. O sistema deve possuir uma faixa de referência configurável (por padrão: 400–600 Hz).
4.2. A decisão final deve indicar se a frequência corresponde ao padrão do Aedes aegypti ou não.
4.3. O algoritmo deve tolerar pequenas variações naturais (ex: diferenças por sexo ou idade do mosquito).
4.4. O sistema deve registrar e disponibilizar a frequência comparada e o resultado da verificação.

5. **O algoritmo deve ignorar sons que não correspondem ao padrão de batida de asas de insetos**

5.1. O algoritmo deve rejeitar sinais com ruído branco ou sons contínuos sem periodicidade.
5.2. A detecção de padrões de repetição (modulação rítmica) deve ser parte do processo de validação.
5.4. O algoritmo deve ter taxa de falso positivo inferior a 10% em presença de sons não relacionados a insetos.

6. **O algoritmo deve carregar a base de dados para comparação das frequências**

6.1. A base de dados deve conter registros com faixas de frequência conhecidas de insetos voadores.
6.2. A base deve ser carregada automaticamente na inicialização do algoritmo.
6.3. A base de dados deve ser extensível e permitir atualização manual ou automatizada.

7. **O algoritmo deve criar um espectrograma a partir da frequência**

7.1. O espectrograma deve exibir a intensidade das frequências ao longo do tempo (tempo vs. frequência vs. amplitude).
7.2. O espectrograma deve ter resolução suficiente para destacar frequências entre 200 e 1000 Hz.
7.3. O espectrograma deve ser gerado automaticamente ao processar o áudio. 7.4. O espectrograma deve destacar ou anotar a frequência identificada como dominante.

Os critérios acima garantem que o algoritmo desenvolvido atenda aos requisitos de precisão, desempenho e confiabilidade esperados pelo projeto DengBuster.

Histórico de ferramentas

Essa seção visa documentar as tecnologias utilizadas para o desenvolvimento do algoritmo de detecção de mosquitos da dengue. Inicialmente foi utilizado o MATLAB para a codificação, posteriormente, após algumas análises com o grupo foi decidido que uma migração de linguagem se fez necessária de MATLAB para Python, pois dessa maneira toda a parte de codificação do projeto estaria em uma única linguagem, não será necessária de uma paga para a execução do código e o consumo de memória diminui pela não necessidade de uma interface do MATLAB.

MATLAB

A MATLAB é uma plataforma de programação e computação numérica usada por milhões de engenheiros e cientistas para analisar dados, desenvolver algoritmos e criar modelos.

Python

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, conhecida por sua sintaxe clara e legível, que favorece a produtividade, a facilidade de manutenção do código e possui uma vasta biblioteca padrão e uma comunidade ativa que contribui com milhares de pacotes e frameworks.

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	01/05/2025	Vinícius de Oliveira, Bruno Ricardo de Menezes

Requisitos

Introdução

Este documento apresenta os **Requisitos Funcionais e Não Funcionais do Algoritmo** de classificação responsável pela detecção do mosquito. Tais requisitos foram definidos com base nas necessidades práticas do sistema embarcado, visando garantir **eficiência, seletividade, resposta em tempo real e operação offline**.

A seguir, são listados os requisitos que guiarão o desenvolvimento e validação do algoritmo de detecção.

Requisitos Funcionais

Nº	Requisito
01	O algoritmo deve classificar o mosquito através da frequência do batimento das asas.
02	O algoritmo deve receber como entrada o sinal de áudio captado por um microfone.
03	O algoritmo deve extrair a frequência do som das asas do mosquito a partir do sinal de áudio.
04	O algoritmo deve comparar a frequência extraída com a faixa de frequência característica do mosquito da dengue (Aedes aegypti).
05	O algoritmo deve ignorar sons que não correspondem ao padrão de batida de asas de insetos.
06	O algoritmo deve carregar a base de dados para comparação das frequências.
07	O algoritmo deve criar um espectrograma a partir da frequência.

Requisitos Não Funcionais

Nº	Requisito
01	O sistema deve funcionar offline, sem necessidade de conexão com a internet.
02	O algoritmo deve processar o sinal de áudio em tempo real, com latência máxima de 2 segundos.
03	O algoritmo de processamento de imagens deve identificar o mosquito da dengue em menos de 2 segundos.

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	01/05/2025	Vinícius de Oliveira, Bruno Ricardo de Menezes

Guia de Instalação e Uso: Detecção de Mosquito da Dengue por Som

Pré-requisitos

- Python 3.10+
- Git

Instalação

Guia de Instalação para o uso do algoritmo de som - **Linux (Ubuntu/Debian)**

1. **Instale o Python (se necessário)**:

```
bash
  sudo apt update
  sudo apt install python3 python3-pip -y
```

1. **Crie um ambiente virtual (opcional, mas recomendado)**:

```
bash
  python3 -m venv venv
  source venv/bin/activate
```

1. **Instale as dependências**:

```
bash
  pip install -r requirements.txt
```

1. **Instale dependências do sistema para `sounddevice`**:

```
bash
  sudo apt install libportaudio2 libsndfile1
```

Guia de Instalação - **Windows**

1. **Instale o Python**:
2. Baixe em: https://www.python.org/downloads/
3. Marque a opção **"Add Python to PATH"** na instalação.
4. **Abra o terminal (CMD ou PowerShell)** e vá até a pasta do projeto:

```
cmd
  cd caminho\para\sua\pasta
```

1. **(Opcional) Crie um ambiente virtual**:

```
cmd
  python -m venv venv
  venv\Scripts\activate
```

1. **Instale as dependências**:

```
cmd
  pip install -r requirements.txt
```

1. **(Se necessário)** instale o driver do `sounddevice` (PortAudio):
2. Baixe o instalador PortAudio se houver erro.
3. Ou use o comando:

```
cmd
 pip install pipwin
```

```
pipwin install sounddevice
```

Após instalação

1. Clone o repositório

```
git clone https://gitlab.com/unb-esw/fga-pi2/semestre-2025-1/dengue/software/algorithm.git
cd algorithm-main/DENGUE_modelo_imagens
```

2. Execução dos algoritmos

Executar o algoritmo criarBaseDados.py com os áudios dos mosquitos no mesmo diretório para criar a base de dados:

```
python criarBaseDados.py
```

Após a criação da base de dados executar o arquivo matching.py para executar o algoritmo que verifica se o mosquito é ou não o Aedes Aegypt:

```
python matching.py
```

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	30/05/2025	Vinícius de Oliveira

Guia de Instalação e Uso: Detecção de Mosquito da Dengue com YOLOv8

Este projeto utiliza o modelo **YOLOv8** para treinar e detectar imagens relacionadas ao mosquito da dengue, com resultados visuais, métricas de performance e vídeos de predição.

Pré-requisitos

- Python 3.10+
- Conda (recomendado)
- Git
- Placa GPU com suporte CUDA (opcional, mas recomendado)

Instalação

1. Clone o repositório

```
git clone https://gitlab.com/unb-esw/fga-pi2/semestre-2025-1/dengue/software/algorithm.git
cd algorithm-main/DENGUE_modelo_imagens
```

2. Crie e ative o ambiente Conda

```
conda env create -f enviroment.yml
conda activate yolov8dengue
```

Alternativamente, instale manualmente os pacotes indicados no `environment.yml`.

3. Instale o Ultralytics YOLOv8

```
pip install ultralytics
```

Ou clone direto do repositório: `git clone https://github.com/ultralytics/ultralytics.git && cd ultralytics && pip install -e .`

Uso

Treinamento do Modelo

Para baixar o dataset e treinar o modelo:

```
python dengue_download_dataset_and_train.py
```

Isso irá:

- Baixar e preparar o dataset
- Iniciar o treinamento com YOLOv8
- Salvar pesos como `best.pt`, `last.pt` etc.

Inferência / Predição

Use o notebook `dengue_deploy.ipynb` para:

- Carregar o modelo (`best-10.pt` ou `best-80.pt`)
- Executar predições em imagens ou vídeos
- Visualizar resultados com bounding boxes

Execute em ambiente Jupyter Notebook:

```
jupyter notebook dengue_deploy.ipynb
```

Histórico de Versão

Versão	Descrição	Data	Responsável
1.0	Criação do documento	30/05/2025	Vinícius de Oliveira