

# Detecção e Classificação de Carros e Placas de Trânsito com FOMO e YOLO

Ricardo Magno do Carmo Junior, 2025100500, Universidade Federal de Itajubá

## CONTEÚDO

<b>I</b>	<b>Introdução</b>	1
	I-A Definição do Problema e Motivação . . . . .	1
<b>II</b>	<b>Dataset e Preparação de Dados</b>	1
	II-A Coleta de Dados . . . . .	1
	II-B Anotação e Processamento . . . . .	3
	II-C Aumento de Dados (Data Augmentation) . . . . .	3
	II-D Balanceamento do Dataset . . . . .	3
<b>III</b>	<b>Arquitetura do Modelo e Treinamento</b>	4
	III-A Modelo 1: FOMO . . . . .	4
	III-B Modelo 2: YOLO NCNN . . . . .	5
<b>IV</b>	<b>Otimização para Implantação Embarrada</b>	6
	IV-A Quantização . . . . .	6
	IV-B Otimização de Pipeline e Resolução . . . . .	6
<b>V</b>	<b>Avaliação de Desempenho e Análise</b>	6
	V-A Precisão (Accuracy) . . . . .	6
	V-B Velocidade de Inferência (FPS) . . . . .	6
	V-C Monitoramento de Recursos . . . . .	7
<b>VI</b>	<b>Desafios na Contagem de Objetos</b>	7
<b>VII</b>	<b>Melhorias Futuras e Extensões</b>	7
<b>VIII</b>	<b>Resultados</b>	7
<b>Apêndice</b>		8
	A Diagrama da Arquitetura do Sistema . . . . .	8
	B Configuração de Hardware . . . . .	8
	C Dependências de Software e Instalação . . . . .	8
	D Repositório . . . . .	8

## I. INTRODUÇÃO

Este projeto, realizado como culminação da Parte 1 do curso, aborda o desenvolvimento de um sistema de Visão Computacional embarcado para a detecção de carros e placas de trânsito. O objetivo principal é implementar uma solução funcional e em tempo real utilizando hardware de baixo custo e restrito, especificamente a **Raspberry Pi Zero 2W**.

A motivação por trás deste projeto é desenvolver uma base para um sistema maior de prevenção de risco no trânsito, e, além disso, o autor precisava de uma desculpa para andar mais de moto. A aplicação prática dos conceitos das Semanas 1-7 em um desafio real de Edge AI é o foco central deste trabalho.

## II. DATASET E PREPARAÇÃO DE DADOS

### A. Coleta de Dados

A criação de um dataset robusto foi o primeiro passo crítico do projeto. Devido à natureza específica da tarefa (captura de carros e placas em um cenário real), optou-se por criar um dataset personalizado.

Foi utilizado um script em Python na Raspberry Pi Zero 2W, configurado para capturar imagens do módulo de câmera a uma taxa de **24 fotos por segundo**. Este método de "burst" permitiu a coleta rápida de dados em diferentes condições de iluminação e ângulos com o objetivo final de criar um vídeo para comparação posterior. O processo resultou em um dataset bruto total de **15.000 imagens**.

Também se fez necessário a criação de um suporte (figura 3) para acoplar a Raspberry no painel da moto em conjunto com uma caixa (figuras 1 e 2) para proteção e melhor locomoção do conjunto Raspberry e câmera. Nesse protótipo, utilizou-se de um powerbank externo para alimentação da placa, mas também era possível acoplar uma bateria ao interior da caixa.



Figura 1. Caixa desenvolvida



Figura 3. Suporte para prender a caixa ao painel da moto (Projeto e soldado por meu pai)



Figura 2. Interno da caixa

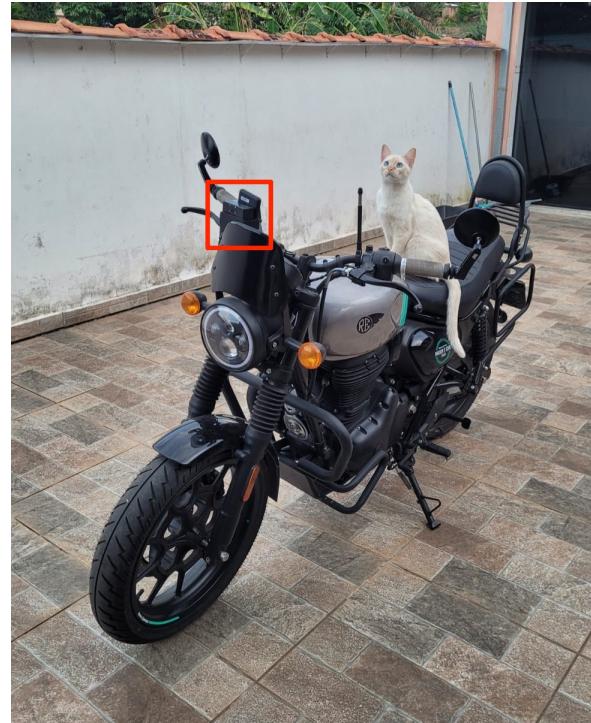


Figura 4. Localização do suporte e caixa na moto (Presença de José Vitorino)



Figura 5. Caixa com a câmera apontada para frente (José Vitorino inspecionando o serviço)



Figura 6. Anotação das classes



Figura 7. Classificação de falso-positivos pelo COCO, como o carro no outdoor

#### B. Anotação e Processamento

Do total de 15.000 imagens, uma sub-amostra de 1/12 (aproximadamente **1.250 imagens**) foi selecionada para o processo de anotação manual. Esta seleção garante que serão utilizadas 2 imagens por segundo e que todo o percurso gravado poderá ser anotado para criar o dataset.

O processo de anotação (criação de \*bounding boxes\*) foi realizado utilizando a plataforma **Roboflow**. As classes definidas para o projeto foram "car" e "sign". Com ajuda da ferramenta de anotação automática (do COCO dataset), o processo de anotação foi agilizado, já que os carros eram facilmente identificados; porém, as placas ainda requeriam serem anotadas manualmente, já que o COCO dataset possui a classe "stop sign" mas que parece ter dificuldades em reconhecer as placas de trânsito brasileiras. Porém, mesmo com a ajuda da anotação automática do Roboflow, ainda deve-se ter atenção aos falso positivos marcados, como por exemplo o carro no outdoor da figura 7.

#### C. Aumento de Dados (Data Augmentation)

Para aumentar a robustez e generalização dos modelos, foi aplicado a técnica de aumento na claridade (brightness) em 15% nas imagens, já que o horário de coleta foi às 17:00 aproximadamente e o dia estava nublado.

#### D. Balanceamento do Dataset

Ao final, Dataset foi separado em 70% treino, 20% validação e 10% teste, com uma contagem total de 2.294 carros e 1.422 placas. A análise total dos dados pode ser observada na figura 8.

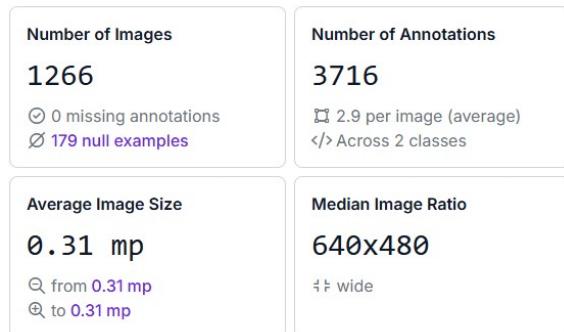


Figura 8. Análise dos dados

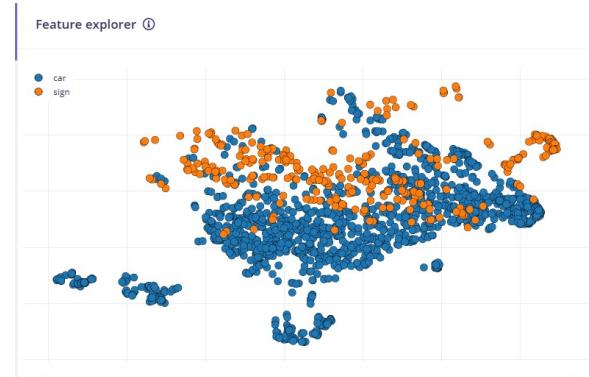


Figura 10. Feature Explorer no EI

### III. ARQUITETURA DO MODELO E TREINAMENTO

Para atender aos requisitos do projeto, foram exploradas duas abordagens distintas de modelagem e treinamento, visando comparar a performance extraída ao final.

Para ambos os modelos, os dados foram carregados das anotações feitas no RoboFlow.

#### A. Modelo 1: FOMO

- Plataforma:** Edge Impulse.
- Arquitetura:** Foi utilizada a arquitetura **FOMO (Faster Objects, More Objects)**, baseada em uma MobileNetV2 0.35 quantizada (INT8), otimizada para detecção de objetos em tempo real em dispositivos com recursos limitados.
- Resolução de Entrada:** 160x160 pixels.
- Treinamento:** O modelo foi treinado na plataforma Edge Impulse por 25 épocas, com uma taxa de aprendizado de 0.001.
- Resultado (Validação):** O modelo atingiu uma precisão (accuracy) de **75%** no conjunto de validação da plataforma.

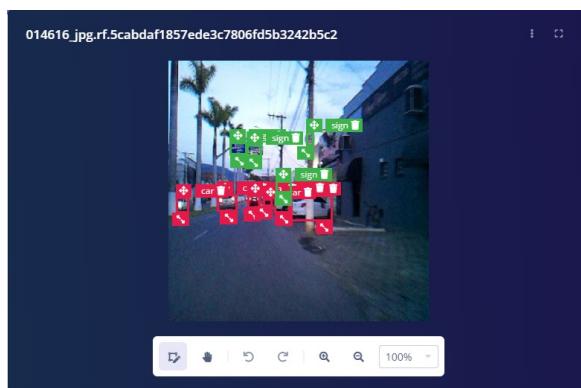


Figura 9. Exemplo de imagem anotada no EI

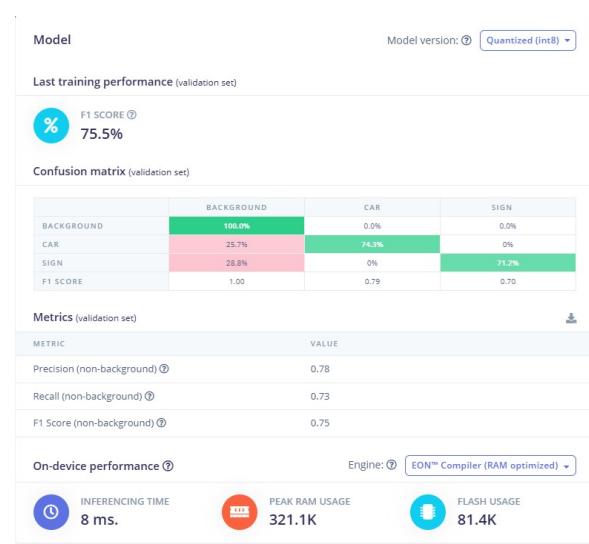


Figura 11. Performance do FOMO no EI



Figura 12. Teste do FOMO no EI

Nota-se aqui uma performance relativa no treinamento do modelo, porém, uma performance péssima no teste do modelo. Evidenciado pela figura 13 onde o

FOMO identifica muitos falso-positivos, especialmente no centro da imagem.

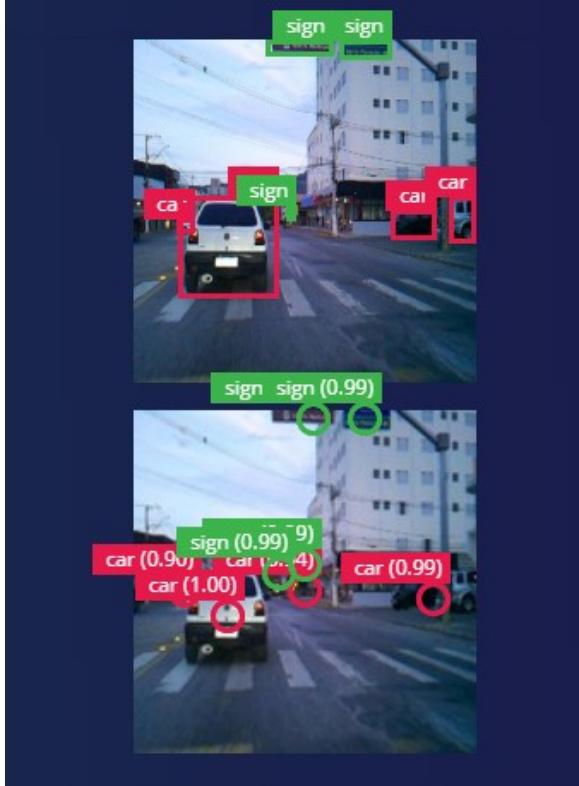


Figura 13. Teste realizado em um frame no EI

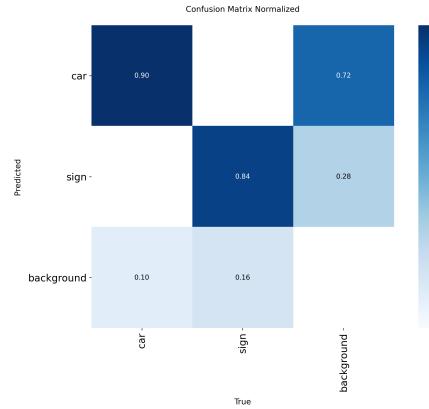


Figura 14. Matriz de confusão do YOLO

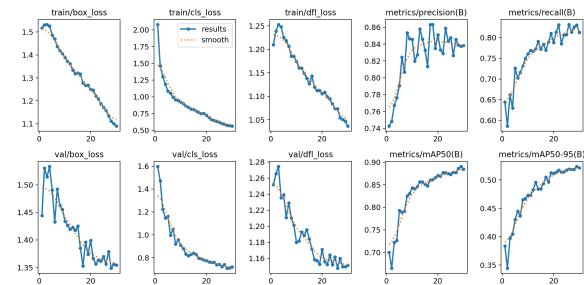


Figura 15. Métricas do YOLO

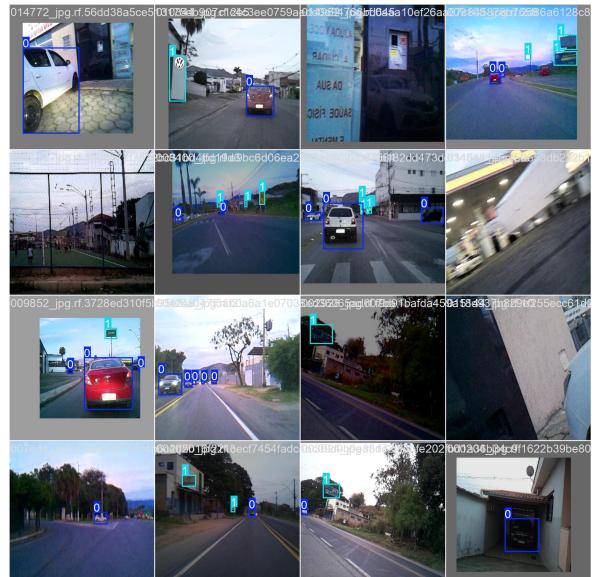


Figura 16. Exemplo de Batch do treinamento

### B. Modelo 2: YOLO NCNN

- Plataforma:** Google Colab (utilizando GPU Tesla T4).
- Arquitetura:** Foi treinada uma variante da **YOLO** (You Only Look Once) YOLO11n. O modelo foi subsequentemente convertido para o formato NCNN.
- Resolução de Entrada:** 640x640 pixels.
- Treinamento:** O modelo foi treinado no Google Colab por 30 épocas, utilizando o otimizador AdamW.
- Resultado (Validação):** O modelo atingiu uma precisão (mAP) de **89,1%** no conjunto de teste.



Figura 17. Exemplo de Batch de validação

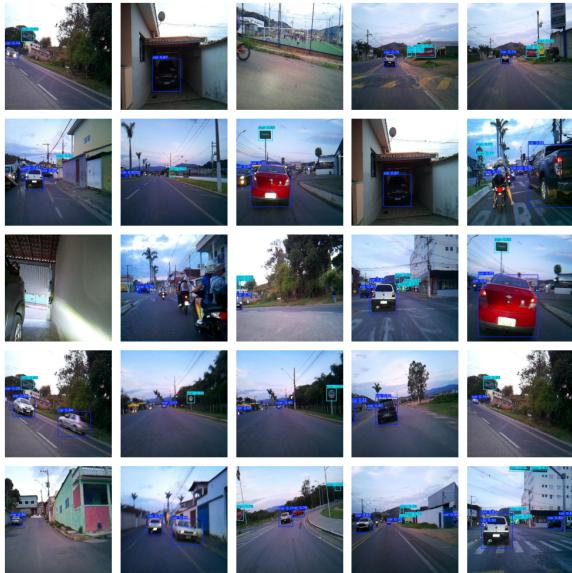


Figura 18. Teste realizado em 25 imagens

A efeito de curiosidade, ambos modelos serão utilizados para classificar os dados coletados anteriormente, com o processamento sendo feito em um computador, somente para efeito de comparação de capacidade. O vídeo comparativo se encontra nos anexos.

#### IV. OTIMIZAÇÃO PARA IMPLANTAÇÃO EMBARCADA

A implantação na Raspberry Pi Zero 2W exige otimizações rigorosas.

##### A. Quantização

A técnica de otimização primária foi a **quantização**. Ambos os modelos foram convertidos para operar com

inteiros de 8 bits (INT8) em vez de ponto flutuante de 32 bits (FP32).

- O modelo FOMO foi quantizado nativamente pela plataforma Edge Impulse.
- O modelo YOLO foi quantizado durante o processo de conversão para NCNN.

Isso reduziu drasticamente o tamanho do modelo e o custo computacional da inferência.

##### B. Otimização de Pipeline e Resolução

A partir da criação destes modelos, analisamos o trade-off entre a resolução de entrada e a performance. Resoluções maiores (como 640x480 para o YOLO) fornecem maior precisão na detecção de objetos pequenos (placas), mas aumentam o tempo de inferência. Resoluções menores (160x160 para o FOMO) são muito mais rápidas, mas podem perder detalhes. O pipeline de pré-processamento (captura, redimensionamento, normalização) foi otimizado usando OpenCV para minimizar a latência. Além da criação de uma Thread especializada no armazenamento das imagens que ficam temporariamente em uma fila na RAM, diminuindo o tempo necessário para salvar; este método não possui influência no pipeline geral do YOLO (já que o tempo de inferência domina o pipeline), mas para o FOMO é crítico na garantia dos 24fps.

#### V. AVALIAÇÃO DE DESEMPENHO E ANÁLISE

Os modelos foram implantados e testados diretamente na Raspberry Pi Zero 2W.

##### A. Precisão (Accuracy)

A precisão no mundo real foi consistente com os resultados de validação:

- **FOMO:** Atingiu 75% de precisão, apesar de apresentar muitos falso-positivos.
- **YOLO NCNN:** Atingiu 87% de precisão, com a impressão de ser melhor que esse número.

O modelo YOLO NCNN demonstrou uma capacidade superior de generalização e detecção de placas menores, provavelmente devido à sua arquitetura mais complexa e maior resolução de entrada.

##### B. Velocidade de Inferência (FPS)

A velocidade foi o principal indicador de desempenho em tempo real. Os testes foram executados medindo o tempo total do loop (captura, pré-processamento, inferência, pós-processamento).

Tabela I  
COMPARAÇÃO DE PERFORMANCE NA RASPBERRY PI ZERO 2W

Modelo	Precisão	Velocidade (FPS)
FOMO (Edge Impulse)	75%	<b>23.56</b>
YOLO NCNN (Colab)	87%	<b>0.9</b>

Ambos os modelos atenderam ao requisito mínimo de 0.5 FPS. O modelo FOMO foi significativamente mais rápido, atingindo com folga 24 FPS e uma média de 23.56fps, enquanto o YOLO NCNN operou a 0.9 FPS devido ao peso do processamento requerido.

### C. Monitoramento de Recursos

O monitoramento foi implementado no canto superior esquerdo das imagens, um exemplo por ser observado nas figuras 19 e 20:



Figura 19. Frame capturado pela Raspberry classificado com FOMO



Figura 20. Frame capturado pela Raspberry classificado com YOLO

## VI. DESAFIOS NA CONTAGEM DE OBJETOS

A contagem de objetos se mostrou um problema neste projeto, por 3 motivos principais:

- Como a câmera está em movimento, ela se torna suscetível às imperfeições do trajeto ou mudanças de ângulo de ataque da motocicleta.
- O ambiente de classificação se torna dinâmico, pois com a motocicleta em movimento, os objetos estarão se movendo ao capturar imagens, mas, com a motocicleta parada, carros podem estar se movimentando (na via oposta por exemplo) enquanto que placas se mantêm paradas.

- Os objetos são suscetíveis à luz ambiente, dependendo do horário do dia, o sol pode atrapalhar a classificação dos objetos e a contagem se perder frame a frame.

Para isso, foi implementado um sistema de rastreamento básico, a posição do objeto é salva e armazenada por 3 segundos (72 frames no FOMO e 3 frames no YOLO), se no próximo frame um objeto da mesma classe é identificado 200 unidades ao redor daquele mesmo objeto, não é contabilizado como uma nova unidade.

## VII. MELHORIAS FUTURAS E EXTENSÕES

Embora o projeto tenha atendido a todos os requisitos, existem várias extensões possíveis:

- **Embutir bateria:** Adicionar uma bateria e módulo de carregamento dentro da caixa para se tornar independente do powerbank.
- **Rastreamento de Objetos:** Implementar algoritmos de rastreamento (ex: Kalman Filter) para seguir os veículos detectados entre os frames.
- **Botões e switch:** Adicionar botões físicos na lateral da caixa para começar e finalizar os scripts de inferência e um switch para ligar ou desligar a alimentação.
- **Otimização de Hardware:** Explorar o uso de câmeras melhores para obter melhor resolução de imagem.
- **Ampliar o Dataset:** Contornar o problema da luz solar ao adicionar mais imagens em diferentes horas do dia.

## VIII. RESULTADOS

Os resultados deste projeto se mostram suficientes para o contexto em que foram aplicados. O uso de FOMO ou YOLO depende muito da especificação e da tarefa que eles se propõem a resolver. Se o projeto precisa de muita velocidade de classificação e pode abrir mão de certa precisão, com certeza o FOMO é o melhor caminho, mas, se o projeto precisa de precisão alta e precisa de poucas imagens, YOLO é o que deve ser escolhido.

Neste projeto, se a priorização fosse criar um vídeo a 24 frames por segundo, a melhor escolha seria classificar por FOMO, porém, se o projeto for somente contar os carros e placas para monitoramento, sem nem mesmo precisar salvar a imagem, YOLO seria uma melhor opção.

O que define o sucesso do modelo aplicado é a escolha do qual melhor se adapta ao problema a ser solucionado, ambas tecnologias de classificação de objetos cumprem o objetivo que propõem a resolver, basta que o projetista saiba usar as ferramentas que tem em mãos.

## APÊNDICE

### A. Diagrama da Arquitetura do Sistema

Os diagramas 21 e 22 ilustra o fluxo de dados do sistema implementado.

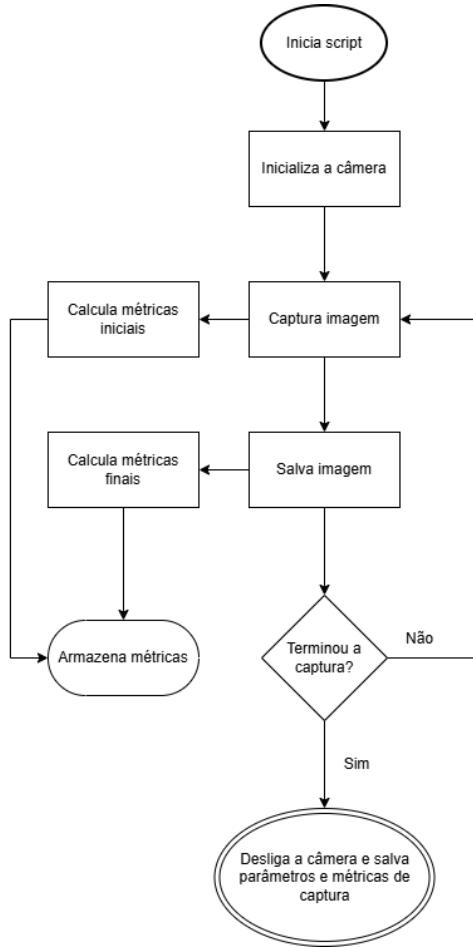


Figura 21. Diagrama de captura de dados

### B. Configuração de Hardware

- **Plataforma:** Raspberry Pi Zero 2W
- **Câmera:** OV5647
- **Fonte de alimentação:** 5V / 2.5A
- **Cartão SD:** 32GB Classe 10

### C. Dependências de Software e Instalação

O sistema foi desenvolvido em **Python 3.9** e utiliza um conjunto de bibliotecas padrão (como `os`, `time`, `threading`, `sys`, `datetime`, `signal`, `math`, `io`, `atexit`) e pacotes de terceiros.

As principais dependências externas e seus comandos de instalação (via `pip`) estão listadas abaixo:

- **Bibliotecas Principais de CV e Numéricas:**
  - **NumPy:** `pip install numpy`
  - **OpenCV (Headless):** `pip install opencv-python-headless`
  - **Pillow (PIL):** `pip install Pillow`

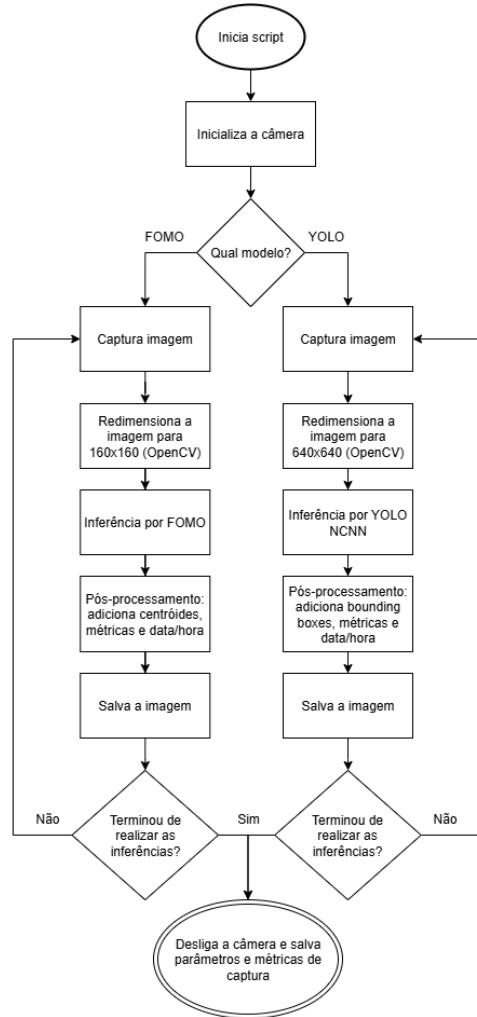


Figura 22. Diagrama de inferência

### • Runtimes de Machine Learning:

- **TensorFlow Lite Runtime:** `pip install tflite-runtime`
- **NCNN:** `pip install ncnn`

### • Hardware e Sistema:

- **Controle da Câmera:** `pip install picamera2`
- **Monitoramento do Sistema:** `pip install psutil`

### • Servidor Web:

- **Flask:** `pip install Flask`

### • Utilitários:

- **Timezones:** `pip install pytz`

### D. Repositório

O código-fonte completo com links para o dataset e vídeos está disponível no repositório: [GitHub](#).