UNIVERSIDADE FEDERAL DE SERGIPE | DEPARTAMENTO DE COMPUTAÇÃO

# FrutoMorph

## Uma Aplicação de Processamento de Imagens para Identificação de Frutas em Python

Wemerson da Silva Soares *wemersonsilvasoares9@gmail.com*
Prof. Leonardo Nogueira Matos *Processamento de Imagens - COMP0432*

*São Cristóvão/SE - 2025*

# Estrutura da Apresentação

- Introdução e Contextualização
- Estrutura Geral do Projeto
- Funcionamento do Algoritmo
- Resultados Obtidos
- Desafios Encontrados e Soluções
- Conclusões e Perspectivas Futuras
- Referências

# I. Introdução e Contextualização

O presente projeto propõe uma solução acessível e eficaz para a identificação de frutas utilizando técnicas de imagens. Sem recorrer a métodos baseados em aprendizado de máquina, que demandam grandes volumes de dados e alto poder computacional, optou-se por uma implementação manual e determinística.

Para isso, foram escolhidas técnicas bem estabelecidas em Processamento de Imagens, como filtros de suavização, a detecção de bordas com o filtro Sobel e a aplicação da Transformada de Hough Circular para identificar formas circulares.

Embora essa abordagem ofereça vantagens, como simplicidade e baixo custo, também apresenta desafios e limitações, especialmente no que diz respeito ao tempo de execução e à menor flexibilidade em cenários complexos, configurando-se como uma verdadeira prova de conceito no escopo do projeto.

# FrutoMorph | Solução Proposta

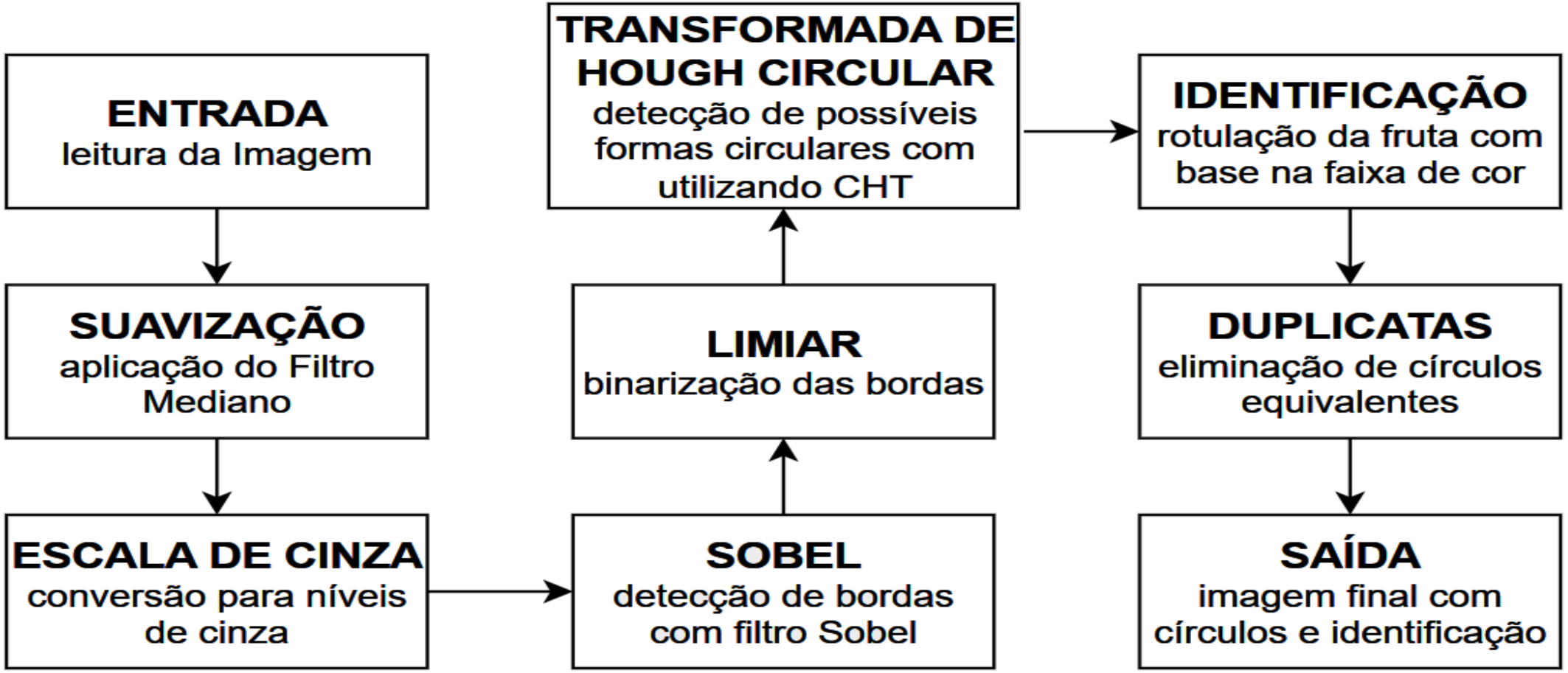
- **Contexto:** A aplicação do processamento de imagens pode se apresentar como uma alternativa ao aprendizado de máquina, especialmente em áreas como a agricultura, onde a automação de processos contribui para a melhoria da eficiência e redução de custos.

- **Justificativa:** A identificação automática de frutas, por exemplo, facilita a separação e a eventual classificação, otimizando processos industriais e auxiliando em tarefas de controle de qualidade.

- **Objetivo:** O objetivo deste trabalho é aplicar exclusivamente técnicas de processamento de imagens para detectar e identificar frutas em imagens, utilizando abordagens como níveis de cinza, filtros de suavização, o operador de Sobel e a Transformada de Hough Circular.

# II. Estrutura Geral do Projeto

O processamento é realizado em várias etapas sequenciais. A seguir, o diagrama esquemático mostra o processo desde a leitura a obtenção da imagem final.



# FrutoMorph | Diagrama Esquemático

→ **Entrada**: A imagem da fruta é carregada para ser processada. Aqui, você pode mostrar como a imagem é lida a partir do arquivo, utilizando a função *imread*.

→ **Suavização**: A imagem é suavizada com um filtro mediano, onde cada pixel recebe o valor mediano de seus vizinhos em uma janela de tamanho específico para remover ruídos e detalhes menores.

→ **Escala de Cinza**: A imagem colorida é convertida para escala de cinza, simplificando a informação visual e focando apenas nas intensidades de luz.

→ **Sobel**: O filtro Sobel é aplicado para detectar as bordas na imagem. Isso é feito por meio de convoluções que calculam o gradiente da imagem em direções específicas, destacando as bordas dos objetos.

→ **Limiar**: As bordas detectadas são binarizadas (preto e branco) para separar as regiões de interesse (as frutas) do fundo, com a aplicação de um limiar.

→ **Transformada de Hough Circular**: Usando a CHT, a imagem é analisada para detectar círculos que podem representar as frutas. A técnica mapeia as bordas para um espaço paramétrico, onde são identificados possíveis centros e raios.

→ **Identificação**: A cor média dentro de cada círculo detectado é calculada e convertida para o espaço de cores HSV. Baseado na tonalidade (hue), saturação e valor, a fruta é identificada (por exemplo, maçã, laranja, etc.).

→ **Duplicatas:** Círculos que estão muito próximos uns dos outros são eliminados para evitar duplicações. A distância mínima entre os centros é definida para garantir que cada fruta seja identificada apenas uma vez.

→ **Saída**: A imagem final é gerada, exibindo as frutas detectadas, circuladas, e rotuladas no centro de cada círculo. Essa imagem é salva e exibida.

# Construção do Dataset | Pré-Processamento das Imagens

Antes de iniciar o processamento propriamente dito, todas as imagens passaram por um processo metódico de padronização, que envolveu:

- **Redimensionamento** para uma altura fixa de **160 pixels**, com o objetivo de evitar imagens pesadas e garantir uniformidade no tamanho e na resolução;
- **Conversão** para o formato **PNG**, a fim de manter um padrão de entrada;
- **Remoção** do **canal alfa** (camada de transparência) após a conversão.

As etapas de padronização foram realizadas fora do ambiente de processamento, garantindo que todas as imagens estivessem no formato, resolução e estrutura adequados, resultando em um **dataset** homogêneo para o processamento posterior. Esse processo assegura um fluxo de trabalho mais limpo, eficiente e controlado, permitindo a execução das etapas do projeto de forma consistente, sem a necessidade de ajustes específicos para diferentes tipos de imagem.

# Construção do Dataset | Imagens Selecionadas

0

1

2

3

4

5

6

7

8

9

# III. Funcionamento do Algoritmo

O FrutoMorph tem como objetivo identificar e classificar frutas em imagens utilizando técnicas de processamento de imagem. No desenvolvimento do projeto, foram utilizadas bibliotecas e ferramentas para manipulação de imagens e processamento de dados.

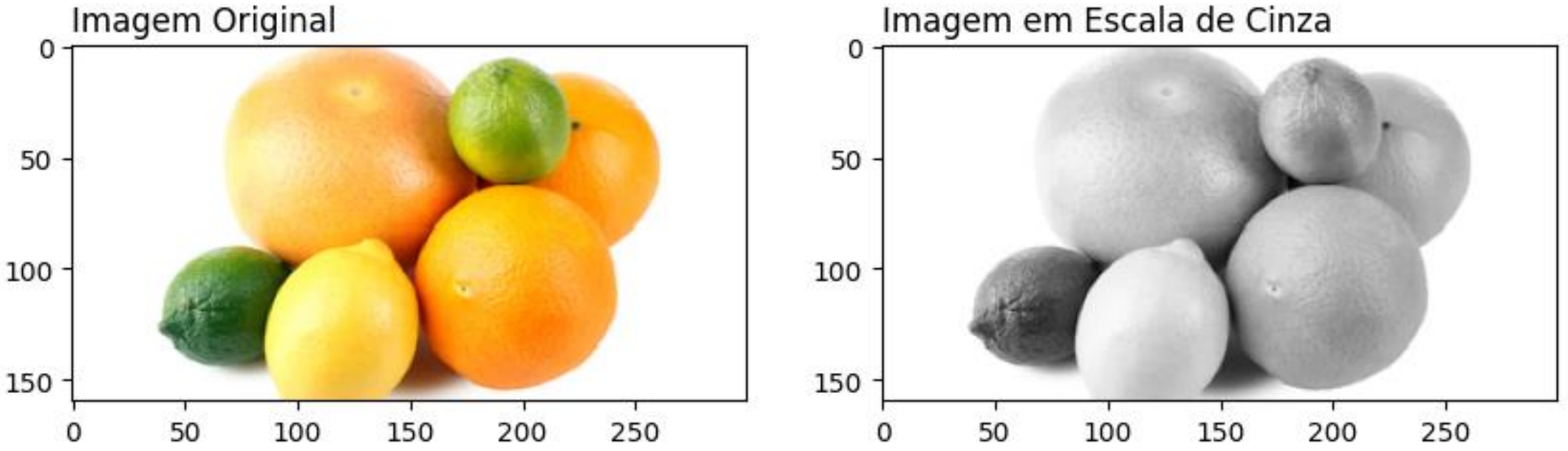
- **Visão Geral das Etapas**: Conforme, visto, o processo de identificação das frutas é ser dividido em várias etapas, que serão descritas a seguir. Cada etapa visa transformar a imagem de uma maneira que facilite a detecção e classificação das frutas.
- **Tecnologias Utilizadas**: A principal biblioteca utilizada no processamento e manipulação de imagens é a ***scikit-image***, que foi rigorosamente aplicada para as operações essenciais do projeto. Além disso, foram empregadas outras bibliotecas auxiliares para exibição de resultados e manipulação de arrays.

# FrutoMorph | Bibliotecas Utilizadas

- **Matplotlib**: Exibição de imagens e gráficos durante o processamento.
- **Matplotlib Image**: Leitura das imagens a partir de arquivos PNG.
- **NumPy**: Manipulação de arrays e operações matemáticas.
- **Skimage (*scikit-image*)**:
  - ***from skimage import color***: Conversão para níveis de cinza.
  - ***from skimage.draw import circle_perimeter***: Desenho das circunferências.
  - ***from skimage.draw import disk***: Indexação de pixels dentro dos círculos.

# Entrada | Leitura e Processamento

O primeiro passo no processamento da imagem é a **Leitura** e **Conversão para Escala de Cinza**, onde a imagem original é carregada com a função ***imread*** e, em seguida, convertida para níveis de cinza usando ***rgb2gray***.

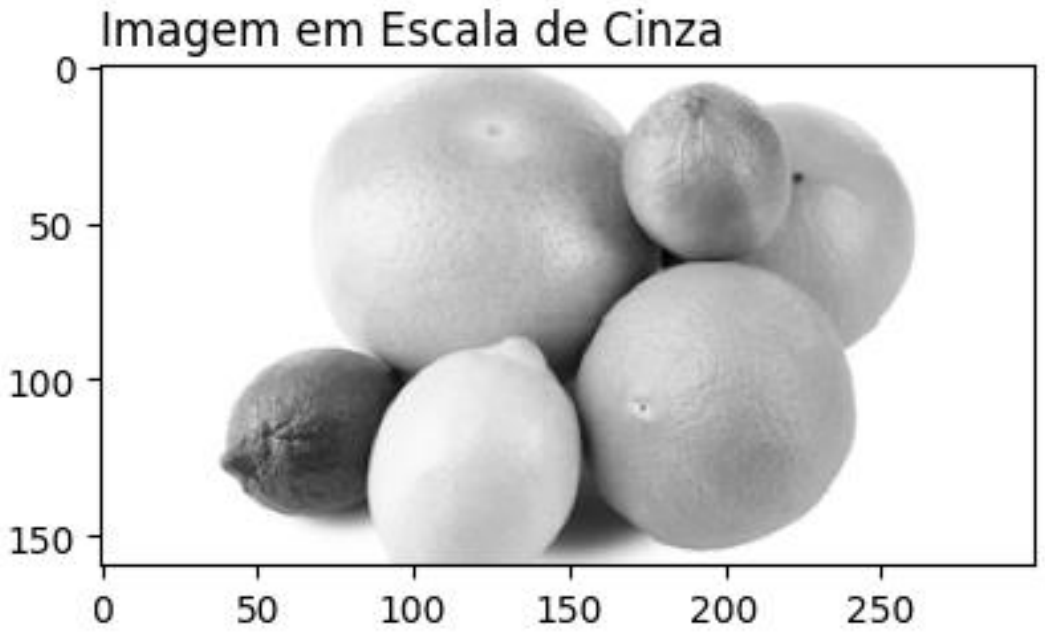


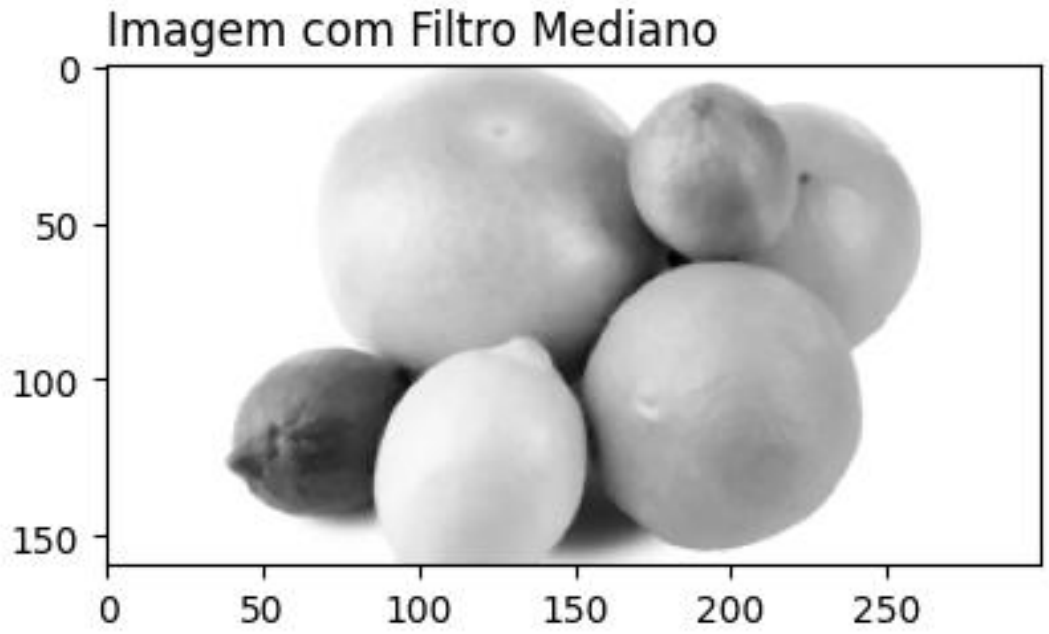

Essa etapa prepara a imagem para o próximo estágio de processamento, tornando-a mais simples e eliminando as informações de cor que não são necessárias para a detecção e análise de formas.

# Suavização | Filtro Mediano

A suavização é realizada utilizando o **Filtro Mediano**, que tem como objetivo reduzir o ruído presente na imagem, tornando a detecção de bordas mais eficaz.

Este filtro preserva as bordas das frutas, enquanto elimina pequenas imperfeições ou distúrbios causados por ruído.



# Suavização | Funcionamento do Filtro Mediano

O **filtro mediano** substitui cada pixel central pela mediana dos valores de seus vizinhos, calculada em uma janela de kernel 3x3. O processo resulta em uma imagem com menos ruído. O exemplo ilustra o funcionamento do filtro mediano, aplicado sobre uma imagem.

- Imagem de Entrada:

$$\begin{bmatrix} 10 & 20 & 30 & 25 & 5 \\ 40 & 15 & 50 & 35 & 60 \\ 25 & 50 & 35 & 10 & 30 \\ 5 & 30 & 10 & 15 & 45 \\ 15 & 20 & 30 & 25 & 10 \end{bmatrix}$$

- Janela de Kernel (3x3) sobre um Pixel Central (15):

$$\begin{bmatrix} 10 & 20 & 30 \\ 40 & 15 & 50 \\ 25 & 50 & 35 \end{bmatrix}$$

- Cálculo da Mediana (ordenação dos valores):

[10 15 20 25 **30** 35 40 50 50]

→ Valor mediano: 30

- Resultado após filtro:

$$\begin{bmatrix} 10 & 20 & 30 & 25 & 5 \\ 40 & \mathbf{30} & 50 & 35 & 60 \\ 25 & 50 & 35 & 10 & 30 \\ 5 & 30 & 10 & 15 & 45 \\ 15 & 20 & 30 & 25 & 10 \end{bmatrix}$$

→ Pixel central alterado para 30

# Detecção de Bordas | Filtro Sobel

O filtro **Sobel** é um operador de detecção de bordas utilizado no processamento de imagens, para destacar as regiões de transição de intensidade, ou seja, as bordas da imagem. Ele é baseado em duas máscaras: uma para detectar bordas na direção **horizontal ($G_X$)** e outra na direção **vertical ($G_Y$)**. O objetivo é calcular o gradiente de intensidade da imagem.

## Magnitude

A Magnitude do gradiente é então calculada para dar uma medida de intensidade da borda.

$$|\text{Gradiente}| = \sqrt{G_x^2 + G_Y^2}$$

## Máscaras

Máscara para a horizontal ($G_X$)

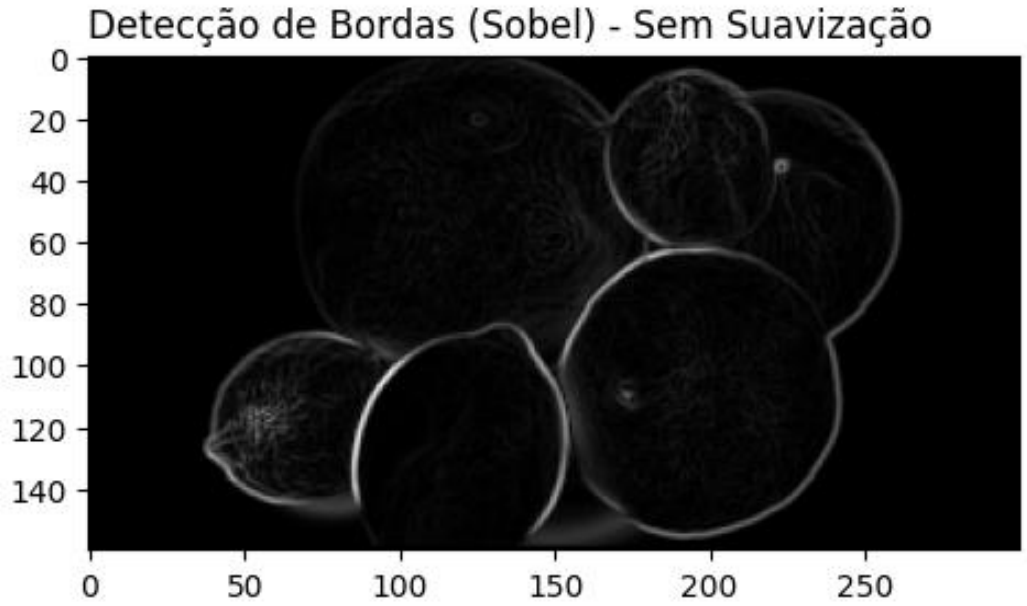
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

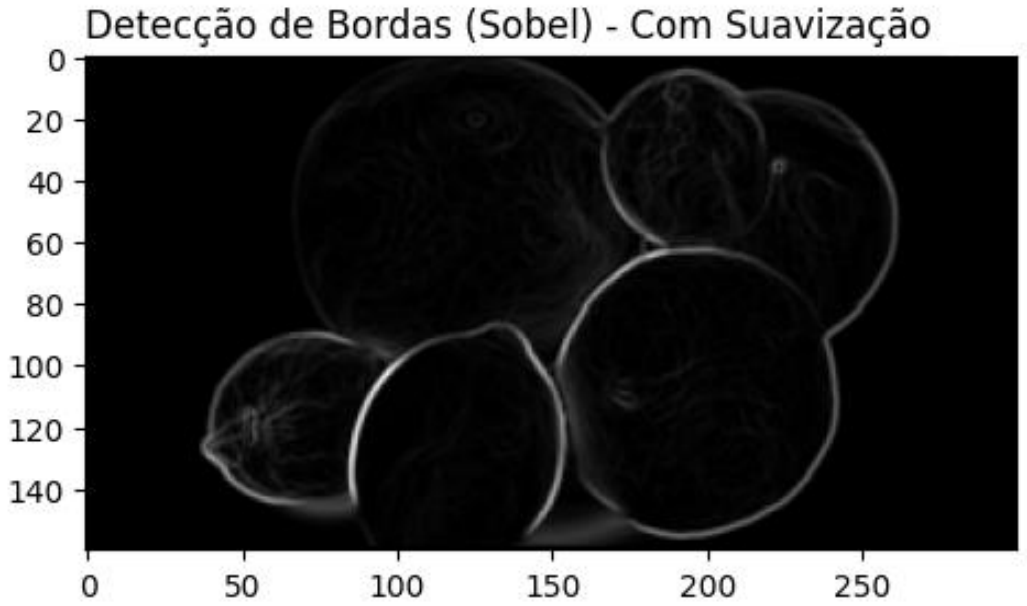
Máscara para a vertical ($G_Y$)

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

# Detecção de Bordas | Aplicação do Filtro Sobel

A **detecção de bordas** usando o **filtro Sobel** é influenciada pela suavização da imagem. Sem a suavização, o filtro Sobel tende a ser mais sensível a ruídos e imperfeições, o que pode resultar em bordas detectadas de forma imprecisa. Quando a suavização é aplicada, como no caso do filtro mediano, as bordas se tornam mais nítidas e menos afetadas por pequenas irregularidades.

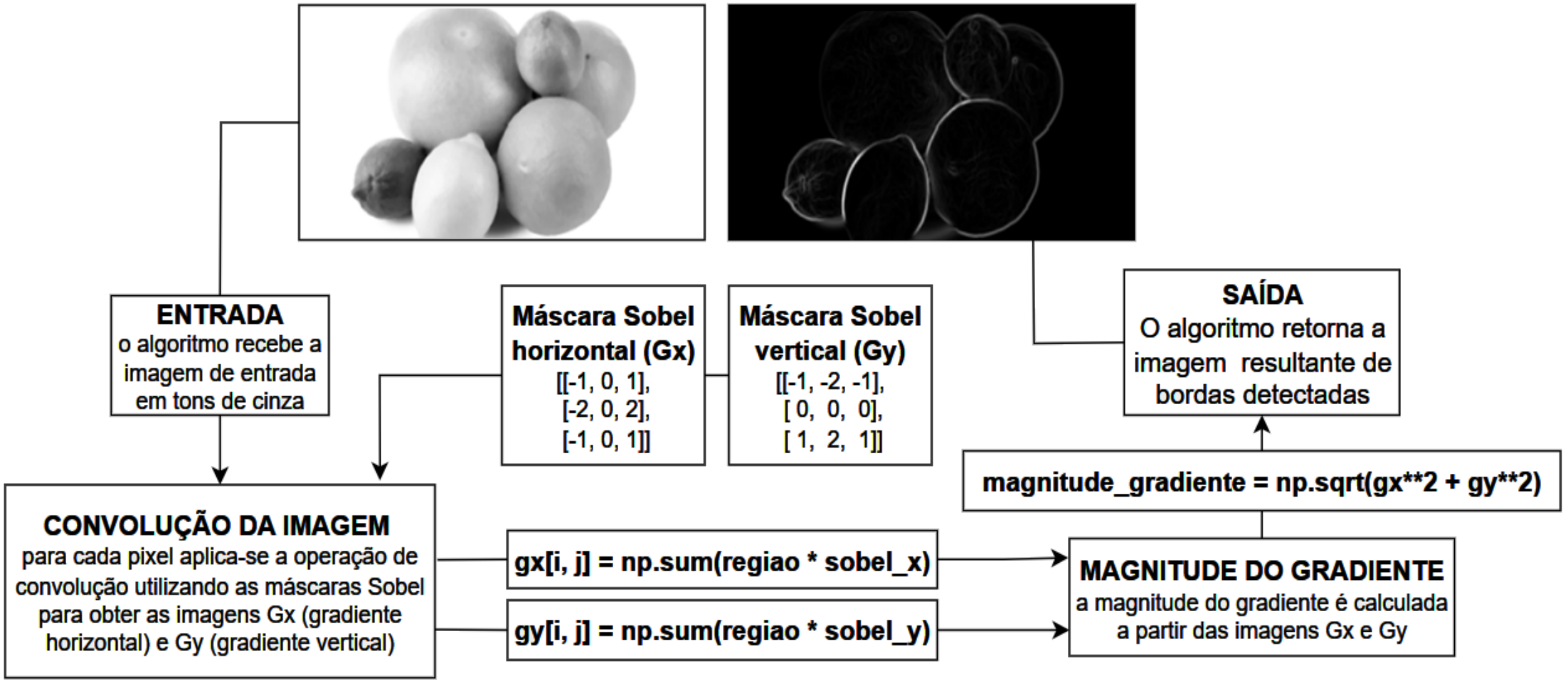



O filtro Sobel detecta essas bordas, destacando, então, os contornos das frutas.

# Detecção de Bordas | Ilustração do Algoritmo

Para ilustrar o fluxo do processo, a seguir é apresentado o diagrama que descreve o funcionamento do algoritmo utilizado para a detecção de bordas.



# Limiar | Binarização

A binarização converte uma imagem em tons de cinza em uma imagem binária, com pixels classificados em **preto (0)** e **branco (255)**. Isso ajuda a destacar elementos importantes, como bordas e objetos.

**Funcionamento**:

Um **valor de limiar** *(threshold)* é definido.

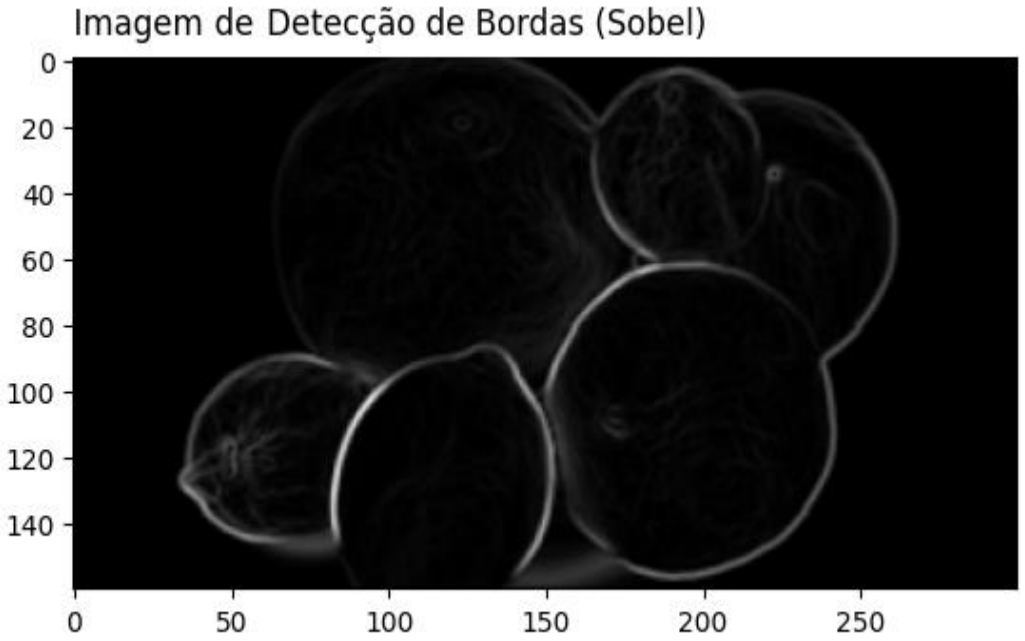
- Se o pixel for maior que o limiar, torna-se **branco (255)**.
- Se o pixel for menor ou igual ao limiar, torna-se **preto (0).**

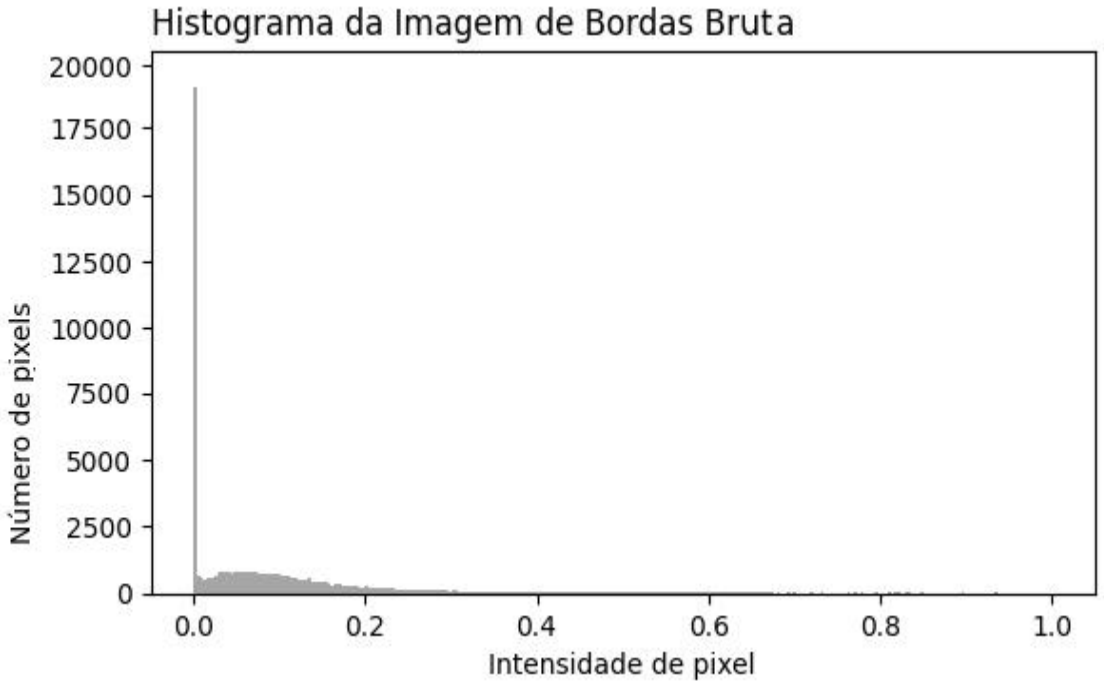
$$I_{bin}(x,y) = \begin{cases} 255 \; se \; I(x,y) > limiar \\ 0 \quad se \; I(x,y) \leq limiar \end{cases}$$

$I(x,y)$ é o valor do pixel original.

$I_{bin}(x,y)$ é o valor do pixel após a binarização.

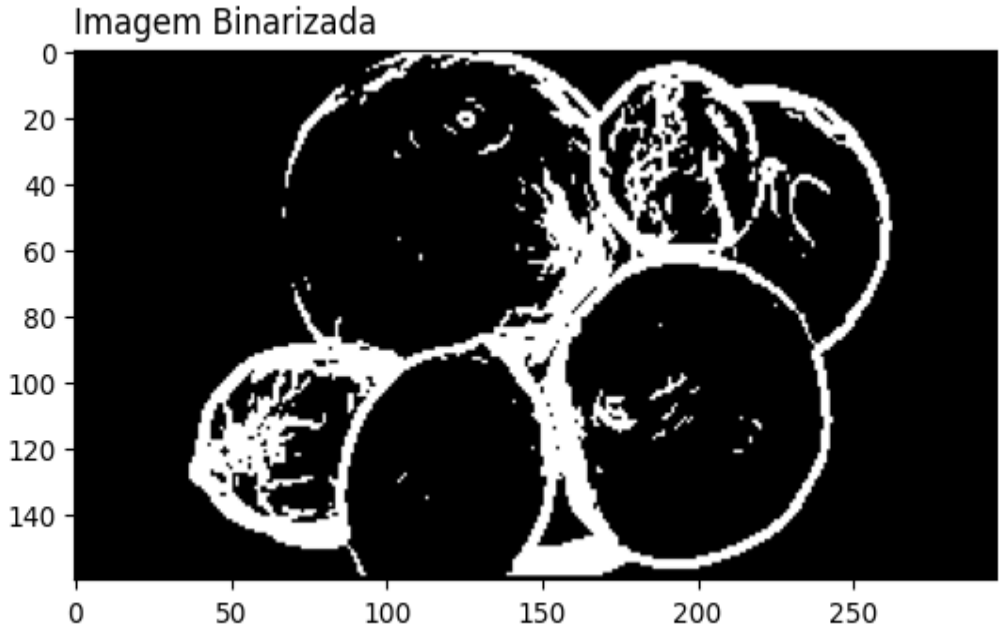
# Limiar | Histograma da Imagem de Bordas

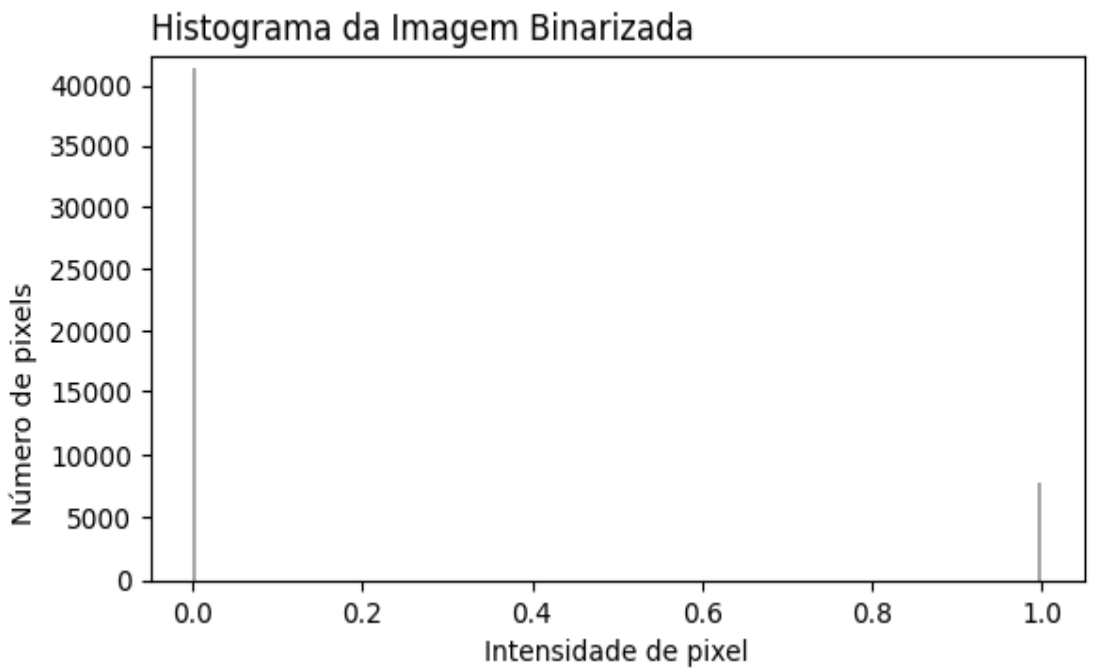



A **imagem de bordas** é uma representação em tons de cinza, onde os pixels têm intensidades variando de 0 (preto) a 1 (branco). O **histograma de sua forma bruta** mostra a distribuição dessas intensidades na imagem, destacando quais valores de pixel são mais predominantes.

# Limiar | Histograma da Imagem Binarizada

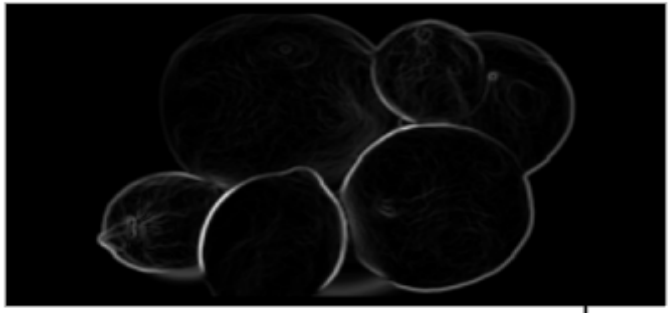


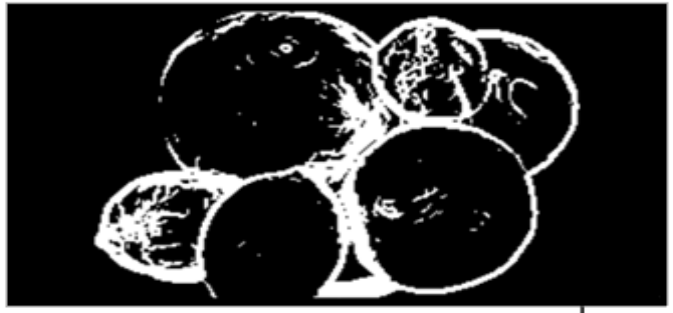




A **imagem binarizada** é uma versão simplificada da anterior, com pixels transformados em preto (0) ou branco (255) com base no limiar. O **histograma binarizado** reflete essa transformação, mostrando dois picos: um para os pixels pretos (0) e outro para os brancos (1), indicando a distribuição binária dos pixels.

# Limiar | Ilustração do Algoritmo

O diagrama a seguir descreve o funcionamento do algoritmo da binarização.





**ENTRADA**
o algoritmo recebe a imagem de bordas detectadas por Sobel

**SAÍDA**
A imagem binarizada (preto 0 e branco 1) é gerada e exibida.

**NORMALIZAÇÃO**
normalizar a imagem para garantir que os valores dos pixels estejam no intervalo [0, 1] caso a imagem não tenha esse intervalo

***imagem_normalizada = imagem / np.max(imagem)***

**DEFINIÇÃO DO LIMIAR**
se o limiar não for fornecido, o limiar é definido como 5% do valor máximo da imagem normalizada.

***limiar=0.05×max(imagem_normalizada)***

**APLICAÇÃO DA BINARIZAÇÃO**
se o valor do pixel for maior que o limiar, o valor do pixel será **1 (branco)**
se o valor do pixel for menor ou igual ao limiar, o valor do pixel será **0 (preto)**

**imagem_binaria[imagem_normalizada > limiar] = 1**

**imagem_binaria[imagem_normalizada ≤ limiar] = 0**

# Transformada de Hugh Circular | CHT

A **Transformada de Hough Circular (CHT)** é uma técnica poderosa em Processamento de Imagens para detecção de formas circulares. O processo envolve três etapas principais: acumulação dos votos para possíveis centros de circunferências, identificação dos centros com maior número de votos e, finalmente, a visualização dos círculos detectados na imagem.

1. **Grade de Acumuladores**: Identifica pixels de borda e acumula votos para possíveis centros de círculos em diferentes raios.
2. **Identificação dos Centros**: Encontra os centros com mais votos, considerando um limiar mínimo.
3. **Desenho dos Círculos**: Desenha os círculos na imagem com base nos centros e raios identificados.

Nos diagramas a seguir, é ilustrado o funcionamento da CHT nas fases de **1** a **3**.

# CHT | Fase 1 - Grade de Acumuladores

**ENTRADA**
o algoritmo recebe a Imagem de Bordas Binária como Entrada

**1 - Grade de Acumuladores**
Identificar os pixels de borda e acumular votos para possíveis centros de círculos.

***pixels_borda = np.argwhere(imagem > 0)***

**Identificação de Pixels de Borda**
processa a imagem binária para encontrar os pixels de borda (onde o valor é maior que zero)

**2 - Cálculo dos Centros dos Círculos**
para cada pixel de borda identificado, calcula-se os possíveis centros para diferentes raios no intervalo de 25 a 60 pixels usando trigonometria (seno e cosseno)

**Cálculo de ângulos**
o círculo é calculado variando os ângulos de 0 a 360 graus para cada pixel de borda.

*#ângulos de 0 a 360 graus*
***thetas = np.arange(0, 360, 1)***

*#coordenada x do centro*
***a = (i - r * np.cos(np.deg2rad(thetas))).astype(int)***

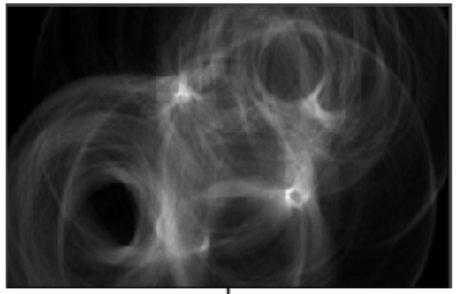
*#coordenada y do centro*
***b = (j - r * np.sin(np.deg2rad(thetas))).astype(int)***

***mascara_valida = (0 <= a) & (a < m) & (0 <= b) & (b < n)***

***a_valido = a[mascara_valida]***

***b_valido = b[mascara_valida]***

**3 - Filtragem de Índices Válidos**
verificar se os centros calculados estão dentro dos limites da imagem (evitar sair da imagem)



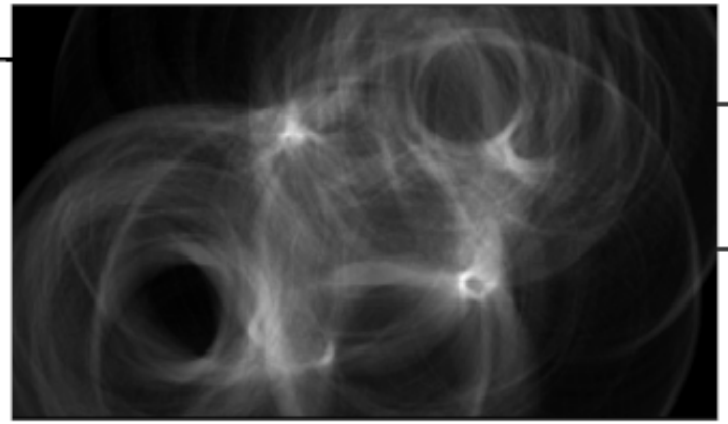
***for ai, bi in zip(a_valido, b_valido):***
***acumulador[ai, bi, r_index] += 1***
*# Incrementa voto no centro (ai, bi)*

**4 - Acumulação de Votos**
após a filtragem, os centros válidos acumulam votos na matriz do acumulador que exibe os centros mais prováveis, com maior intensidade indicando mais votos

**SAÍDA**
grade de acumuladores

# CHT | Fase 2 - Identificação dos Centros

**ENTRADA**
grade de acumuladores



***intervalo_raio***

***limiar = 0.99***

**IDENTIFICAÇÃO DOS CENTROS**
encontrar os centros com mais votos, considerando um limiar mínimo

**BUSCA PELOS MAIS VOTADOS**
após a acumulação, percorre-se a matriz acumuladora para identificar os pontos com o maior número de votos, ou seja, os centros mais fortes

```
max_acum = np.max(acumulador[:, :, r_index])  #máximo número de votos
if max_acum >= limiar * max_acum:
pontos_maximos = np.argwhere(acumulador[:, :, r_index] == max_acum)
```

**FLTRAGEM**
filtra-se apenas os centros que têm votos acima de um limiar mínimo (99% do máximo), para garantir que apenas os centros mais prováveis sejam selecionados

```
#adiciona centro válido à lista
centros.append((x, y, intervalo_raio[r_index]))
```

**SAÍDA**
posição do centro e raio do círculo detectado
`(x, y, raio)`

# CHT | Fase 3 - Desenho dos Círculos

**ENTRADA**
o algoritmo recebe a imagem original para servir de plano de fundo

**COORDENADAS**
posição do centro e raio doe cada círculo detectado
`(x, y, raio)`

***centros = encontrar_centros_circulos (acumulador, intervalo_raio, limiar=0.99)***

**PERÍMETRO**
cálculo do perímetro do círculo
circle_perimeter(x, y, r)

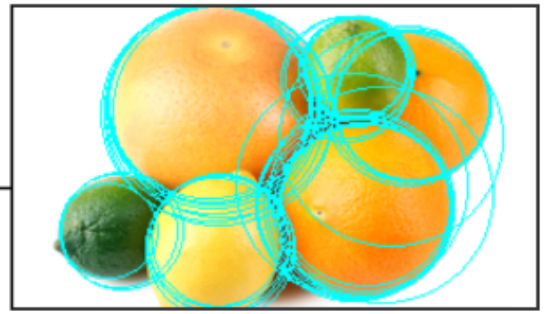
```
from skimage.draw import circle_perimeter
rr, cc = circle_perimeter(x, y, r, shape=imagem.shape)
```

**ÍNDICES VÁLIDOS**
utiliza-se np.clip() para garantir que os índices dos pontos do perímetro estão dentro da imagem

```
#garantir que as coordenadas de linha estão dentro da imagem
rr = np.clip(rr, 0, imagem.shape[0] - 1)
#garantir que as coordenadas de coluna estão dentro da imagem
cc = np.clip(rr, 0, imagem.shape[0] - 1)
```

**DESENHO**
os pontos do perímetro calculados são desenhados na imagem original, marcando os pontos do círculo com uma cor específica (turquesa, RGB [0, 255, 255])

```
#desenhar o círculo turquesa na imagem
imagem_com_circulos[rr, cc] = [0, 255, 255]
```



**SAÍDA**
imagem original com os círculos desenhados, demarcando as áreas detectadas com base nos centros e raios identificados

```
plt.imshow(imagem_com_circulos)
plt.show()
```

# Duplicatas | Eliminação de Círculos Próximos

Durante a detecção das frutas, múltiplos círculos podem ser detectados para o mesmo objeto, especialmente se os objetos estão próximos. Isso gera duplicatas e afeta a precisão da contagem.

A função ***eliminar_circulos_proximos*** remove círculos muito próximos, mantendo apenas um para objetos próximos. Isso evita duplicatas e melhora a precisão na detecção e contagem de objetos.

A proximidade entre os círculos é verificada usando a fórmula da distância euclidiana entre os centros dos círculos:

- **Círculos Detectados**:

$$\begin{cases} \text{Círculo } \mathbf{1}\text{: Centro } (x_1, y_1\,), \text{ Raio } r_1 \\ \text{Círculo } \mathbf{2}\text{: Centro } (x_2, y_2), \text{ Raio } r_2 \end{cases}$$

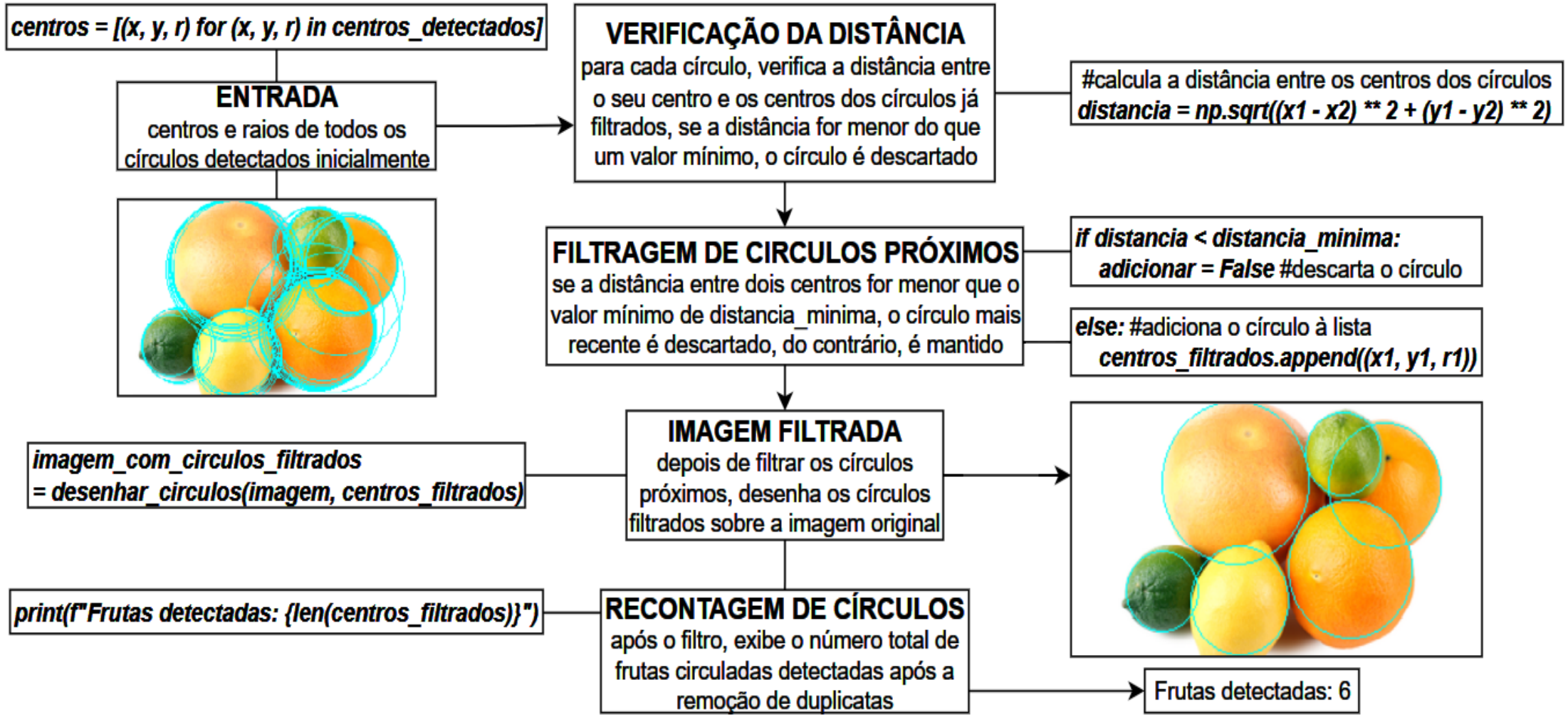
- **Distância Entre Círculos**:

$$\boldsymbol{d} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Se $d < 35(distância\ mínima)$, o Círculo 2 é descartado.

# Duplicatas | Ilustração do Algoritmo

O diagrama a seguir ilustra a eliminação de duplicatas no algoritmo.



# Identificação da Fruta | Cálculo da Cor e Rotulação

Após a detecção dos círculos e a eliminação de duplicatas, o próximo passo é a identificação das frutas presentes na imagem. Esse processo ocorre em duas etapas principais: Calcula-se a cor média da área dentro de cada círculo, fornecendo uma indicação visual da fruta, e em seguida, a cor média é convertida para o modelo de cor HSV, permitindo uma análise mais precisa da matiz (*hue*), saturação (*saturation*) e luminosidade (*value*), que são comparados com intervalos pré-definidos para identificar a fruta. As etapas são as seguintes:

**Cálculo da Cor Média**: Determina-se a cor média da área do círculo, considerando todos os pixels dentro da região.

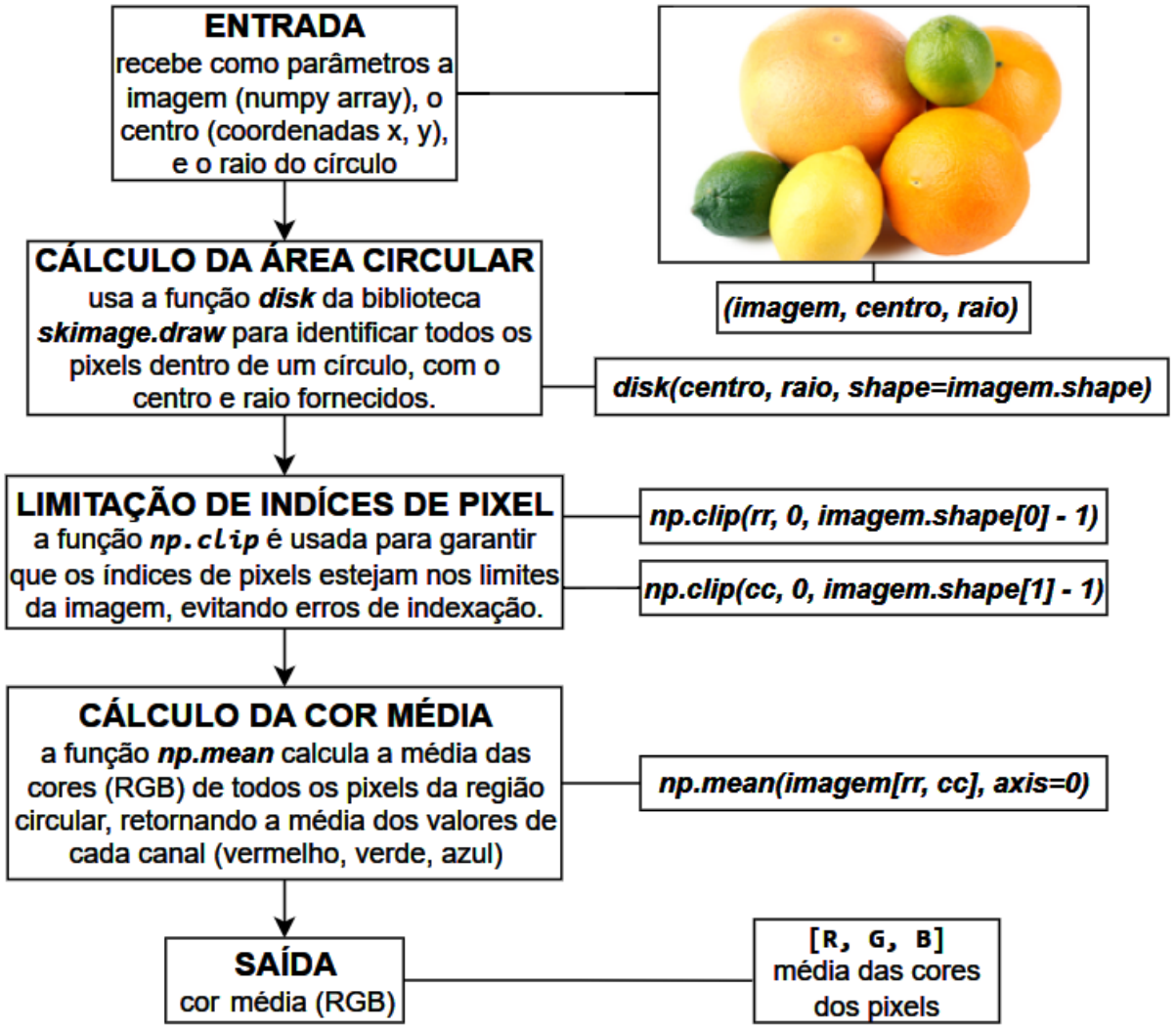
**Análise de Cor com Modelo HSV**: A cor média calculada é convertida para o modelo HSV, e a análise do matiz e saturação permite classificar a fruta com base nos intervalos de cores predefinidos.

**Rotulação das Frutas na Imagem**: Os círculos detectados são desenhados sobre a imagem, e a fruta identificada é rotulada, favorecendo a visualização.

# Identificação da Fruta | Cálculo da Cor Média

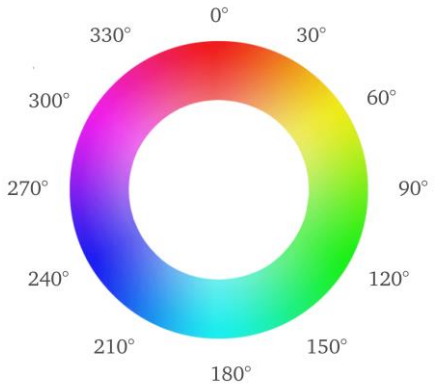
A função calcula a cor média de todos os pixels dentro de uma área circular da imagem, utilizando o centro e o raio do círculo para determinar a região a ser analisada. A média das cores RGB dessa região é então calculada e retornada.

A ilustração ao lado apresenta as etapas do processo de cálculo da cor média, desde a seleção da área circular até o cálculo final da média RGB.



# Identificação da Fruta | Análise de Cor com Modelo HSV

A análise de cor complementa a análise morfológica das frutas, oferecendo uma abordagem simples para estimar a fruta com base em suas características cromáticas. Utilizando o modelo de cor **HSV** (matiz, saturação, valor), é possível distinguir tonalidades de forma mais eficiente que o modelo **RGB**.

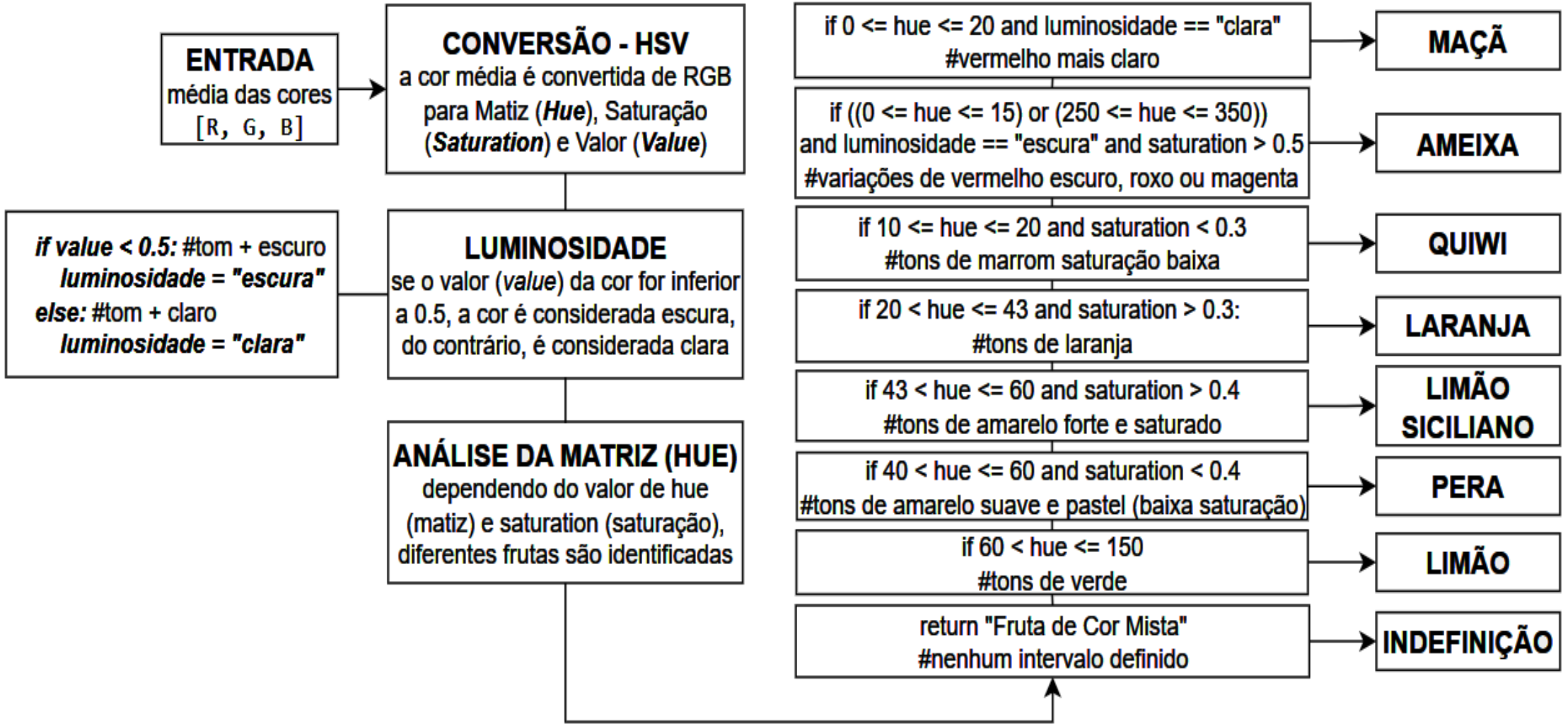



O modelo HSV é composto por três componentes:


- **Matiz (*Hue*)**: Tonalidade da cor, variando de 0° a 360°.
- **Saturação (*Saturation*)**: Intensidade da cor, de 0 (cinza) a 1 (cor pura).
- **Valor (*Value*)**: Luminosidade da cor, de 0 (preto) a 1 (branco).

A cor média da fruta é calculada e, com base na matiz e luminosidade, a fruta é classificada. Esse método oferece uma solução prática para o reconhecimento de frutas, sem necessidade de redes neurais ou aprendizado de máquina.

# Identificação da Fruta | Ilustração da Análise de Cor



# Identificação da Fruta | Rotulação das Frutas na Imagem

A etapa de rotulação vem após a análise da cor média. Nessa fase, os círculos desenhados sobre a imagem são usados para identificar e rotular as frutas com base nas cores médias calculadas. O processo é o seguinte:

1. **Entrada**: **Imagem original** e as **coordenadas dos círculos** (determinadas na etapa anterior).
2. **Desenho dos Círculos**: Círculos são desenhados nas demarcando as frutas.
3. **Rotulação**: Para cada círculo, é calculada a **cor média** da região interna. Com base na **matiz** e **luminosidade**, a fruta é **identificada** e **rotulada**.


**Saída**: A imagem resultante exibe os **círculos** sobre as frutas, com os **rótulos** indicando suas identificações (***detecção**: "Laranja", "Limão", "Laranja", "Limão", "Limão", "Siciliano", "Laranja"*).

# Identificação da Fruta | Ilustração da Rotulação das Frutas

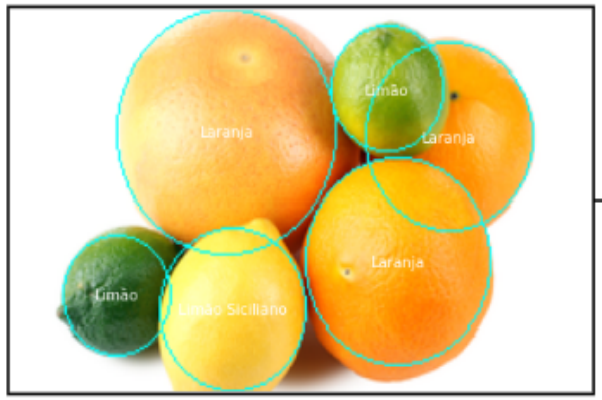
O diagrama a seguir ilustra a rotulação das frutas na imagem.

**CENTROS E RAIOS**
lista de centros detectados (e seus raios) onde os círculos devem ser desenhados

**CÓPIA DA IMAGEM ORIGINAL**
a imagem original é copiada para a variável ***imagem_com_circulos***, permitindo que o código faça as modificações sem alterar o arquivo original

***imagem_com_circulos = imagem.copy()***

**ENTRADA**
imagem original e centros dos círculos




**SAÍDA**
imagem com círculos e identificação das frutas

**DESENHO DOS CÍRCULOS**
a função percorre cada de círculo em ***centros***, calcula a posição do círculo (com base nas coordenadas (***x***, ***y***) e raio *r*) e desenha a circunferência na cópia da imagem (***imagem_com_circulos***), usando ***circle_perimeter***

***rr, cc = circle_perimeter(x, y, r, shape=imagem.shape)***

***imagem_com_circulos[rr, cc] = [0, 255, 255]***

**CÁLCULO DA COR E IDENTIFICAÇÃO**
a cor média é calculada após o desenho dos círculos, que servem apenas como marcadores visuais, e é usada para identificar a fruta

***cor_media = calcular_cor_media(imagem, (x, y), r)***

***fruta = identificar_fruta(cor_media)***

**RÓTULO DA FRUTA**
depois de identificar a fruta com base na cor, a função escreve o nome da fruta no centro do círculo desenhado, utilizando ***plt.text()***

***plt.text(y, x, fruta, color="white", fontsize=8, ha="center", va="center")***

# IV. Resultados Obtidos

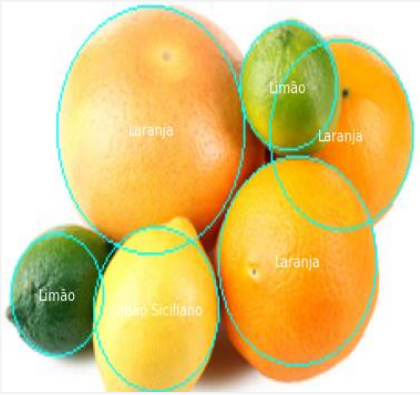
Nesta seção, serão analisados os resultados obtidos a partir da aplicação do algoritmo de detecção de frutas em um conjunto composto por 10 imagens. O objetivo é avaliar a eficácia da detecção, considerando as frutas identificadas, o número de detecções e a precisão do algoritmo em diferentes cenários.

Para cada imagem de entrada, referenciadas na seção *"Construção do Dataset | Imagens Selecionadas"*, será apresentado o resultado da detecção, acompanhado de uma análise detalhada do desempenho, com base na precisão da detecção, no número de frutas corretamente identificadas e nos possíveis erros de classificação.

A seguir, serão apresentadas as imagens do dataset, seguida da análise dos resultados obtidos e do desempenho de cada imagem processada.

# Resultados de Detecção | Imagens Processadas

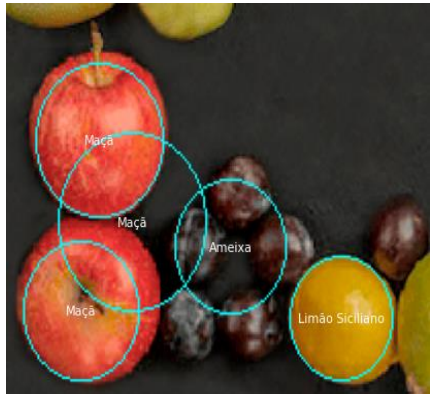
#0



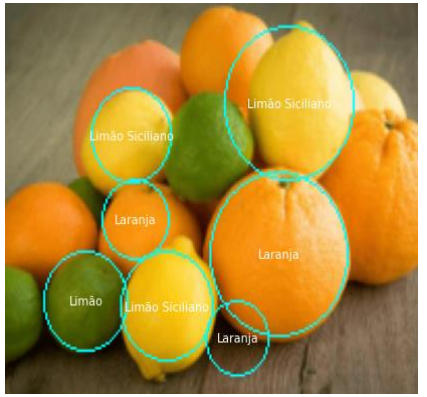

#1


#2


#3




#4

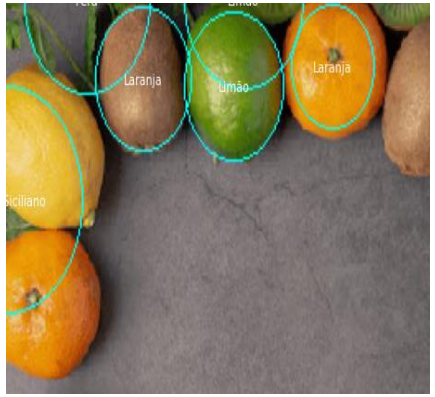



#5

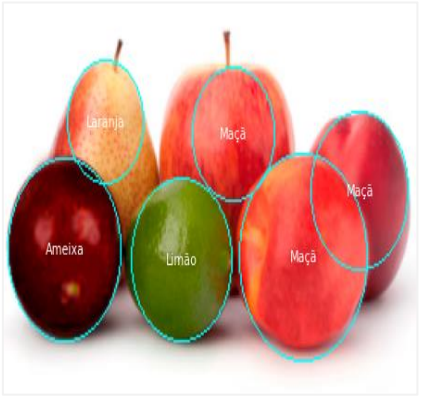

#6


#7


#8

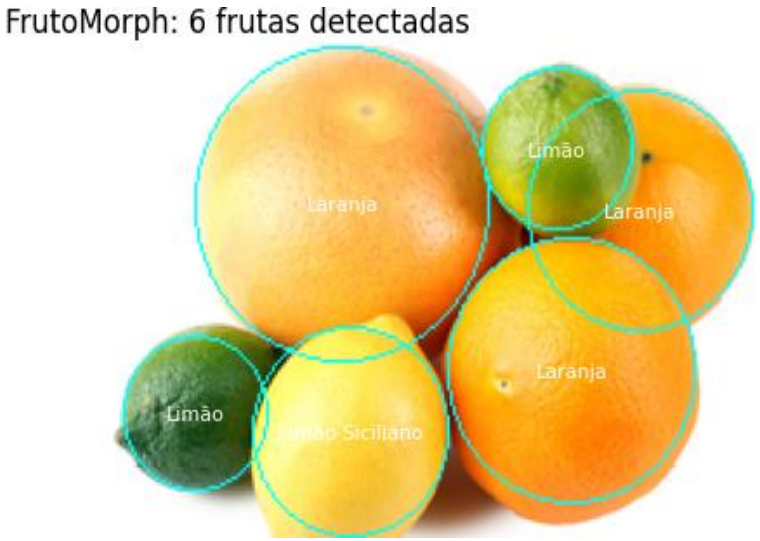



#9



# Resultados | *detecção#0*

A imagem da **Detecção #0** foi escolhida como referência ao longo de toda a explicação no escopo deste texto, pois se trata de uma imagem **clara**, **limpa** e mais **simples**, com **cores vivas**. Seu fundo **completamente branco** e as formas **bem definidas** facilitam a detecção e a análise visual. Para esta imagem inicial, o algoritmo **FrutoMorph** obteve uma **precisão absoluta**.

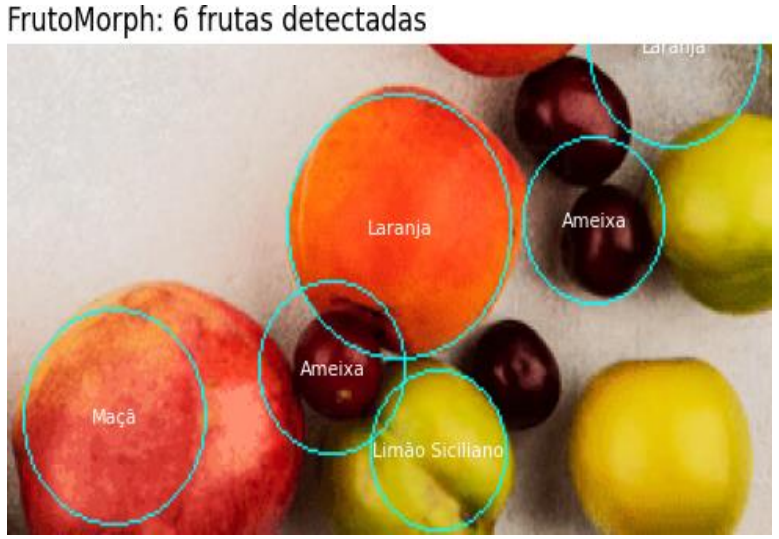



| total | detecções | acertos |
|---|---|---|
| 6 | 6 | 6 |

Foram detectadas **todas** as **6 frutas**, identificando corretamente cada uma delas com base na **análise da cor**. A imagem contém **3 laranjas**, 2 limões verdes e **1 limão siciliano**, e o algoritmo conseguiu classificar corretamente todas as frutas presentes. A precisão do algoritmo pode ser considerada **ideal** para esta imagem, uma vez que **não houve falhas** na detecção ou identificação das frutas.

# Resultados | *detecção#1*

A imagem da **Detecção #1** apresenta maior **diversidade** de circunferências em comparação à imagem anterior, o que representa um **desafio adicional** para o algoritmo. A imagem contém **1 maçã**, **1 laranja**, **4 ameixas** e **3 frutas** cuja cor remete ao **limão siciliano**. Foram identificadas **2 laranjas** (apesar de uma **fora do lugar**), **1 maçã**, **2 ameixas** e **1 limão siciliano** (6/10 frutas).




| total | detecções | Acertos |
|---|---|---|
| 10 | 6 | 5 |

No entanto, deixou **escapar duas ameixas** e **duas frutas** com a mesma cor do **limão siciliano**. Embora o resultado seja satisfatório, essa falha revela uma **limitação** do algoritmo em identificar frutas com **cores semelhantes**, o que foi **intencionalmente testado**. Isso sugere que, embora o algoritmo tenha bom desempenho, ele ainda enfrenta desafios ao lidar com **variações sutis de cor**.

# Resultados | *detecção#2*

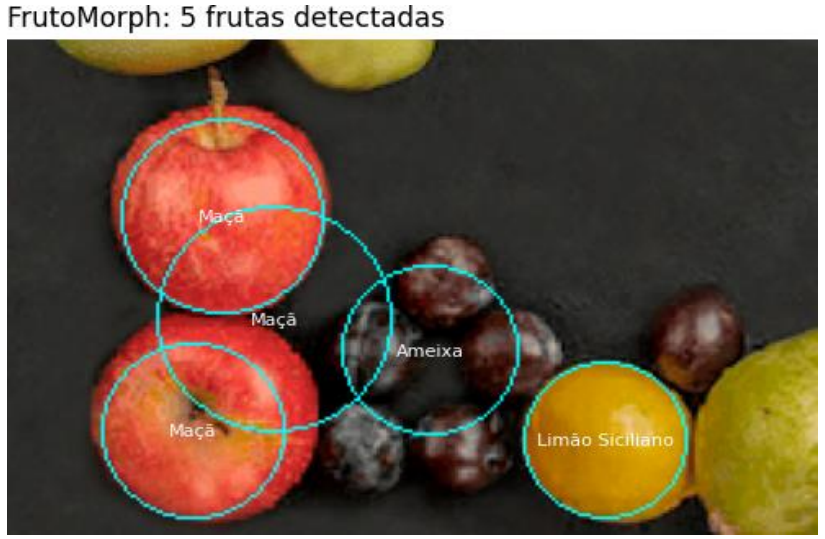
A imagem da **Detecção #2** contém **3 laranjas**, **2 maçãs** e **3 frutas** que se revelaram **problemáticas** durante a detecção: **2 kiwis** e **1 pera**. O algoritmo identificou corretamente as **3 laranjas**, **1 maçã** e a única **pera** (6/8 frutas), mas com algumas ressalvas. Embora a pera tenha sido detectada, sua rotulação foi **equivocada** pela **semelhança** com a cor do limão siciliano.


| total | detecções | acertos |
|---|---|---|
| 8 | 5 | 4 |

Evidencia-se, portanto, a **sensibilidade** do algoritmo a **diferenças sutis de cor**. Uma **maçã** encoberta foi **ignorada**, provavelmente pela dificuldade em detectar seu formato. Quanto aos **kiwis**, o algoritmo não os reconheceu, possivelmente devido à sua **configuração de cor** específica **amarronzada** e ao **formato ovaloide**, que ainda representam um **desafio consistente** para o sistema.

# Resultados | *detecção#3*

A imagem da **Detecção #3** contém **2 maçãs**, **6 ameixas**, **um limão siciliano** e **uma fruta verde circular**. O algoritmo detectou **3 maçãs**, **uma ameixa** e **um limão siciliano** (5/9 frutas). A imagem possui um **fundo mais escuro** e **acinzentado** em comparação com as anteriores. As ameixas são representadas por **círculos menores** em relação às maçãs.




| total | detecções | acertos |
|---|---|---|
| 10 | 5 | 4 |

Possivelmente devido à **disposição** das ameixas, o algoritmo interpretou algumas delas como uma única circunferência, acertando na análise de cor ao identificar corretamente **uma ameixa** na posição central das 5, mas **deixando uma** isolada **escapar**. Quanto à **terceira maçã** identificada, trata-se de um **falso positivo**, resultante da **sobreposição** com áreas vermelhas das maçãs **reais**.

# Resultados | *detecção#4*

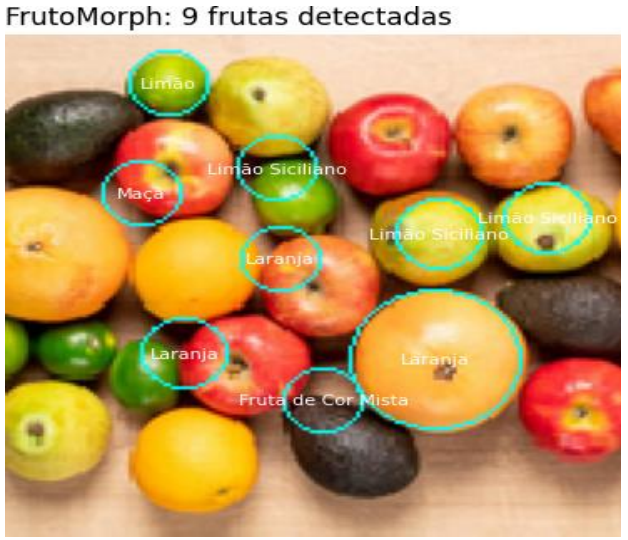
A imagem da **Detecção #4** apresenta uma **quantidade** significativamente **maior de frutas** em comparação com as anteriores, totalizando **6 laranjas**, **3 limões** e **3 limões sicilianos** visivelmente identificáveis. O algoritmo conseguiu detectar **3 laranjas** (sendo uma **um falso positivo**), **3 limões sicilianos** corretamente reconhecidos e **1 limão** verde.


| total | detecções | acertos |
|---|---|---|
| 12 | 7 | 6 |

A **sobreposição** de frutas neste cenário possivelmente representou **um dos maiores desafios** para a detecção das formas. Além disso, a **superfície texturizada** de madeira pode ter contribuído para a ocorrência do falso positivo. No geral, o **FrutoMorph** demonstrou um **bom desempenho** ao distinguir as frutas detectadas, apesar dos **desafios** apresentados pela cena.

# Resultados | *detecção#5*

A imagem da **Detecção #5** apresenta um número igualmente elevado de frutas, incluindo uma espécie não identificada, inserida intencionalmente para **testar falsos positivos**. No entanto, este cenário conta com uma **melhor disposição** das frutas sobre uma superfície plana de tom pastel. No total, a cena contém **6 maçãs**, **3 laranjas**, **4 peras**, **5 limões** verdes e **3 frutas de cor escura**. O algoritmo identificou **3 peras**, embora rotuladas equivocadamente como limão siciliano.

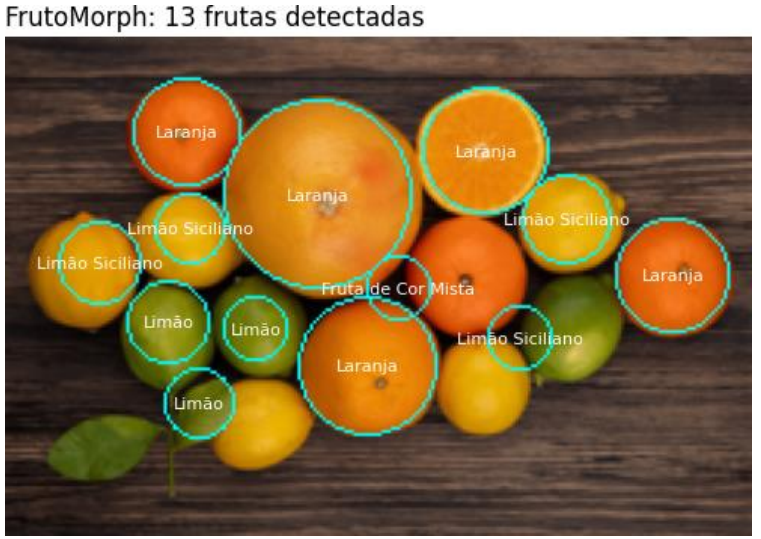



| total | Detecções | acertos |
|---|---|---|
| 21 | 9 | 7 |

Identificou também **1 maçã**, **3 laranjas** (com **um falso positivo**), **1 limão** e **1 das frutas de cor escura**, corretamente classificada como ***"fruta de cor mista"***, abordagem adotada para **idenfinições**. Apesar do maior número de frutas, o algoritmo identificou metade delas, com um nível considerável de precisão.

# Resultados | *detecção#6*

A imagem da **Detecção #6** apresenta uma superfície de madeira de cor **forte**, **escura** e **texturizada**, mas as frutas estão **dispostas** de maneira mais favorável, sem sobreposição e respeitando a vizinhança. A cena contém **6 laranjas**, **5 limões sicilianos** e **3 limões verdes**. O algoritmo identificou todas as **5 laranjas** corretamente, um **sucesso absoluto**, e **4 limões sicilianos** identificados corretamente.

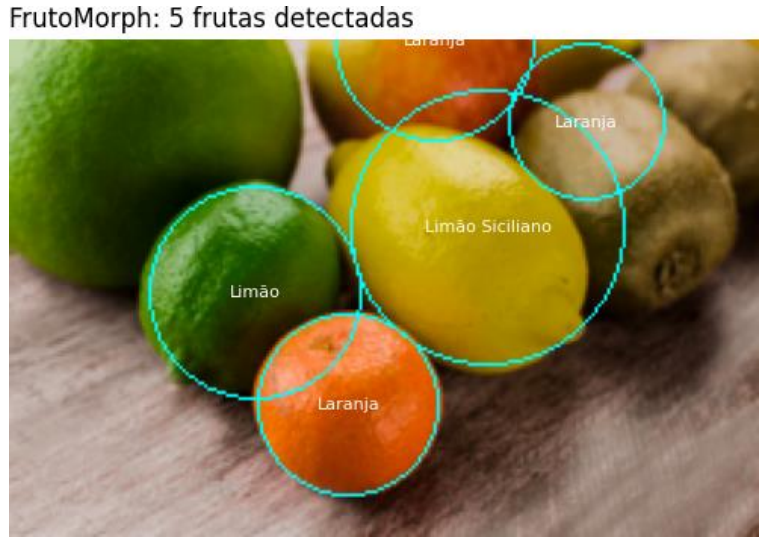



| total | detecções | acertos |
|---|---|---|
| 14 | 13 | 11 |

Quanto aos **2 limões verdes**, devidamente localizados, houve um **falso positivo (**uma folha), e **um limão perdido** pelo seu formato **irregular**. Também houve **falso positivo** no encontro de três laranjas, interpretado erroneamente. Apesar da superfície desfavorável e da **vulnerabilidade** a **falsos positivos** devido às formas, o **FrutoMorph** obteve novamente uma taxa elevada de acertos.

# Resultados | *detecção#7*

A imagem da **Detecção #7** apresenta um número mais contido de frutas, e desta vez, com **tamanhos maiores** em relação às proporções da imagem. Com um total de **2 laranjas, 1 limão siciliano, 1 kiwi e 1 laranja esverdeada**, o algoritmo apresentou os seguintes resultados: **Ignorou a laranja maior**, provavelmente porque o tamanho de seu raio **extrapola** os **limites** estabelecidos nos parâmetros da **Transformada de Hough**.




| total | detecções | acertos |
|---|---|---|
| 6 | 5 | 4 |

Encontrou **um falso positivo para laranja**, onde na verdade deveria ser um **kiwi**, possivelmente devido à configuração específica e amarronzada da casca da fruta, que causou o erro de detecção. **Acertou todas as demais frutas na imagem**, incluindo as **laranjas menores**, o **limão siciliano** e o **kiwi**. Apesar do erro em relação ao **falso positivo**, o algoritmo conseguiu **identificar corretamente a maioria** das frutas, mantendo um **bom índice** de **acertos**.

# Resultados | *detecção#8*

A **imagem da Detecção #8** apresenta uma disposição de frutas deslocadas para as bordas da imagem, com um fundo acinzentado e de textura limpa. Esse cenário favoreceu a detecção de bordas, ao mesmo tempo que testa uma nova configuração para a posição das frutas.


| total | detecções | acertos |
|---|---|---|
| 6 | 6 | 3 |

A cena contém **2 laranjas**, **um limão** verde, **um limão siciliano** e **2 kiwis**.

O algoritmo foi capaz de identificar corretamente **1 limão siciliano** e **1 laranja**, mas cometeu alguns erros: identificou uma **pera (falso positivo)**, sobrepôs **1 limão falso** a **um limão real**, e rotulou o **kiwi** como **laranja**, provavelmente devido à dificuldade em lidar com o **tom amarronzado** da fruta. O algoritmo obteve **50% de acertos**, o que era esperado, considerando as vulnerabilidades do modelo em identificar frutas com **cores semelhantes** ou **sobreposições**.

# Resultados | *detecção#9*

A imagem da **Detecção #9** é a segunda imagem considerada ideal para o algoritmo **FrutoMorph**: o fundo é completamente branco, as frutas estão dispostas de forma sequencial e com um nível de exposição adequado.


| total | detecções | acertos |
|---|---|---|
| 6 | 6 | 5 |

Do total de **3 maçãs, 1 ameixa, 1 limão e 1 pera**, o algoritmo foi capaz de detectar todas as frutas corretamente, com exceção da **pera**, rotulada como **laranja** devido ao seu tom mais alaranjado que os parâmetros definidos no algoritmo.

De modo geral, o algoritmo alcançou um dos melhores desempenhos em termos de detecção, e o único erro de rotulação pode ser atribuído, sem dúvidas, às limitações do determinismo do código, que depende de parâmetros predefinidos para análise das cores.

# Resultados | Desempenho Geral

Para uma visão geral dos resultados, a tabela a seguir sintetiza o desempenho do algoritmo em cada imagem, identificando falhas e acertos. Este panorama possibilita uma avaliação objetiva do modelo nos diferentes cenários analisados.

| dataset' | total | detecções | frutas identificadas | precisão |
|---|---|---|---|---|
| *frutas0* | 6 | 6 | 3 laranjas, 1 limão siciliano, 2 limões verdes | 6/6 = 100% |
| *frutas1* | 10 | 6 | 2 ameixas, 2 laranjas, 1 limão siciliano, 1 maçã | 5/10 = 50% |
| *frutas2* | 8 | 5 | 3 laranjas, 1 limão siciliano, 1 maçã | 4/8 = 50% |
| *frutas3* | 10 | 5 | 1 ameixa, 1 limão siciliano, 3 maçãs | 4/10 = 40% |
| *frutas4* | 12 | 7 | 3 laranjas, 1 limão verde, 3 limões sicilianos | 6/12 = 50% |
| *frutas5* | 21 | 9 | 3 laranjas, 1 limão verde, 3 limões sicilianos, 1 maçã, 1 mista | 7/21 ≈ 33% |
| *frutas6* | 14 | 13 | 5 laranjas, 5 limões sicilianos, 3 limões verdes | 11/14 ≈ 79% |
| *frutas7* | 6 | 5 | 3 laranjas, 1 limão siciliano, 1 limão verde | 4/6 ≈ 67% |
| *frutas8* | 6 | 6 | 2 laranjas, 2 limões, 1 limão siciliano, 1 pera | 3/6 = 50% |
| *frutas9* | 6 | 6 | 1 ameixa, 1 laranja, 1 limão verde, 3 maçãs | 5/6 ≈ 83% |

# V. Desafios Encontrados e Soluções

Durante a análise dos resultados, foram identificadas limitações que impactaram a precisão do algoritmo na detecção e classificação das frutas. Esses desafios estão relacionados tanto a características intrínsecas das imagens quanto a restrições do próprio modelo adotado. A seguir, os principais obstáculos observados ao longo do experimento.

- **Sobreposição de frutas** – Dificuldade na distinção de profundidade e na identificação precisa das formas circulares.
- **Número elevado de frutas** – A redução da escala das frutas em relação à resolução da imagem pode resultar na exclusão de algumas delas do intervalo definido para a Transformada de Hough.
- **Falsos positivos** – Detecção incorreta de elementos não pertencentes ao conjunto de frutas.

# Desafios e Soluções | Desafios

• **Superfícies texturizadas e coloridas** – Insensíveis ao filtro mediano, essas superfícies podem contribuir para a ocorrência de falsos positivos.

• **Diferenças sutis de cores e formas entre frutas** – A rotulação é prejudicada em casos de frutas com coloração semelhante, limitação imposta pelos critérios da matriz HSV (saturação e iluminação). O kiwi não foi rotulado em nenhum dos casos, e a pera, frequentemente confundida com o limão siciliano.

• **Alto número de circunferências sobrepostas ou próximas** – O cálculo da grade de acumuladores com um intervalo consideravelmente amplo para abranger variações de tamanho pode gerar múltiplas detecções para uma mesma fruta ou regiões adjacentes.

Para mitigar essas limitações e aprimorar o desempenho do algoritmo, foram exploradas algumas estratégias e ajustes, conforme detalhado a seguir.

# Desafios e Soluções | Soluções

Com base nos desafios identificados, foram implementadas soluções específicas para melhorar a eficácia do algoritmo, como descrito a seguir.

1. **Matriz Hue (HSV em vez de RGB)** – Para melhorar a distinção entre frutas de cores semelhantes, a análise de cor foi baseada na matiz (*hue*) do espaço de cores HSV, permitindo uma segmentação mais robusta.

2. **Filtro Mediano** – Aplicado para reduzir ruídos e suavizar a imagem antes da detecção de bordas, ajudando a minimizar falsos positivos em superfícies texturizadas.

3. **Filtro Sobel** – Utilizado para detectar bordas mais definidas, melhorando a segmentação das frutas e destacando seus contornos.

4. **Binarização** – Ajustando um limiar para destacar regiões de interesse, reduzindo interferências indesejadas antes da Transformada de Hough.

# Desafios e Soluções | Soluções

**5. Transformada de Hough Circular (CHT)** – Para detectar formatos circulares das frutas, ajustando um intervalo adequado de raios.

**6. Eliminação de Círculos Duplicados** – Implementada para remover múltiplas detecções no mesmo centro ou em regiões muito próximas, reduzindo redundâncias na contagem de frutas.

**7. Ajuste do Limiar na Detecção de Centros** – Para evitar falsos positivos, apenas pontos com votos acima de um percentual do máximo na matriz acumuladora foram considerados.

**8. Análise de Cor Média Dentro do Círculo** – Para rotular corretamente cada fruta detectada, utilizando a média de cores dentro da região identificada.

Essas abordagens foram implementadas de para aprimorar a precisão da detecção e classificação das frutas no conjunto de imagens analisado.

# VI. Conclusões e Perspectivas Futuras

O projeto demonstrou a eficácia de técnicas clássicas de processamento de imagens, como a transformada de Hough e a detecção de bordas, na identificação de frutas em imagens. No entanto, desafios como a sobreposição de frutas, variações sutis de cor e texturas complexas ainda podem impactar a precisão do algoritmo.

Abordagens como o filtro mediano e a análise de cor no espaço HSV ajudaram a mitigar algumas dessas limitações, mas há espaço para aprimoramentos, especialmente na classificação de frutas com cores semelhantes.

No futuro, melhorias podem ser alcançadas com ajustes nos parâmetros do algoritmo, como a definição mais precisa de intervalos de cor e o refinamento da detecção de bordas, além da implementação de técnicas de filtragem mais robustas para lidar com imagens com sobreposição ou textura complexa.

# VI. Conclusões e Perspectivas Futuras

O desempenho também pode ser otimizado por meio da integração do código Python com C. A linguagem C, por ser compilada, oferece vantagens em termos de velocidade de execução, se comparada a Python, uma linguagem interpretada, especialmente em operações de processamento de imagens intensivas.

A implementação de partes do código em C pode reduzir significativamente o tempo de execução, conforme abordado em aula, melhorando a eficiência do sistema em cenários mais desafiadores.

O contínuo aprimoramento dos métodos de processamento e a busca por soluções mais rápidas e precisas garantirão uma maior robustez nas detecções e ampliarão a aplicabilidade do algoritmo em cenários mais diversos.

# VII. Referências


GONZALEZ, R.; WOODS, R. *Digital Image Processing*. 3. ed. Upper Saddle River: Prentice Hall, 2010.

PEDRINI, H.; SCHWARTZ, W. R. *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações*. São Paulo: Thomson, 2007.

SOARES, Wemerson. *FrutoMorph: Diagrama Esquemático*. Diagrama eletrônico no draw.io, 2025. Disponível em: https://shre.ink/FrutoMorph. Acesso em: 30 mar. 2025.

SOARES, Wemerson. *Processamento de Imagens: FrutoMorph*. Repositório no GitHub, 2025. Disponível em: https://github.com/serenesinister/jupyter. Acesso em: 29 mar. 2025.