Contagem de Hemácias

Gabriel Silva de Jesus
Gustavo Henrique Aragão Silva

Docente: Dr. Leonardo Nogueira Matos

Agenda

Problemática e Objetivos

Metodologia

Desenvolvimento

Extração Leucócitos (Isolar Hemácias - HSV)

Pré-processamento: Conversão para Níveis de Cinza, Equalização Histogrâmica e Filtro Gaussiano

Adaptive Threshold

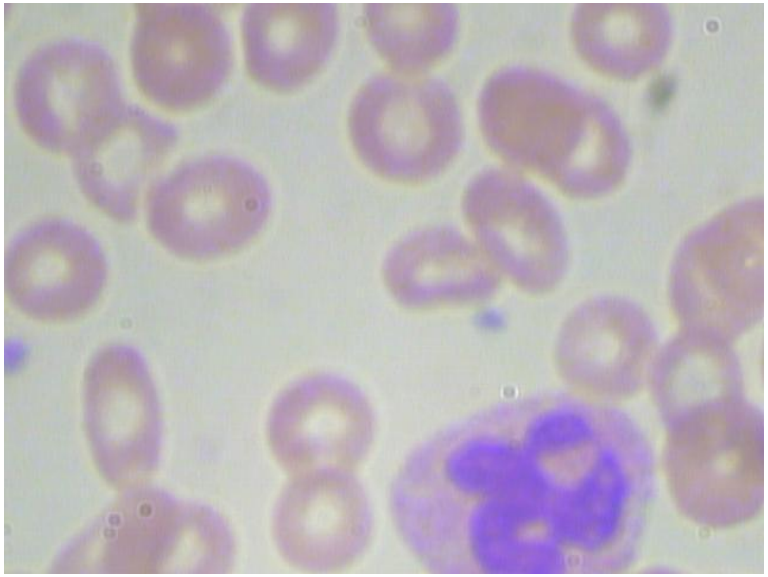
Metodologia da detecção de círculos

Análise de Dados e Conclusão

Referências

Problemática e Objetivos

- O projeto tem como objetivo central desenvolver a **segmentação** de imagens de hemogramas de um dataset a fim de efetuar a **contagem de hemácias** presentes nelas.



Uma das imagens usadas: BloodImage_00408.jpg

Problemática e Objetivos

- Esse processamento é efetuado a partir de *técnicas* aprendidas em sala de aula durante a grade de *Processamento de Imagens*, como **Equalização Histogrâmica**, **Filtros Passa-baixas (Filtro Gaussiano)**, **Morfologia Matemática Binária** e **Transformação de Hough Circular**.
- Entretanto, durante a elaboração do projeto, a fim de um melhor resultado, utilizou-se uma técnica de limiarização e uma de extração de cores não abordada durante as aulas: a **Adaptive Threshold (Limiarização Adaptativa) e Modelo HSV**, que serão melhores discutidos futuramente na apresentação.

Metodologia

Durante a elaboração do projeto, utilizou-se o seguinte *dataset* disponibilizado/recomendado pelo professor:

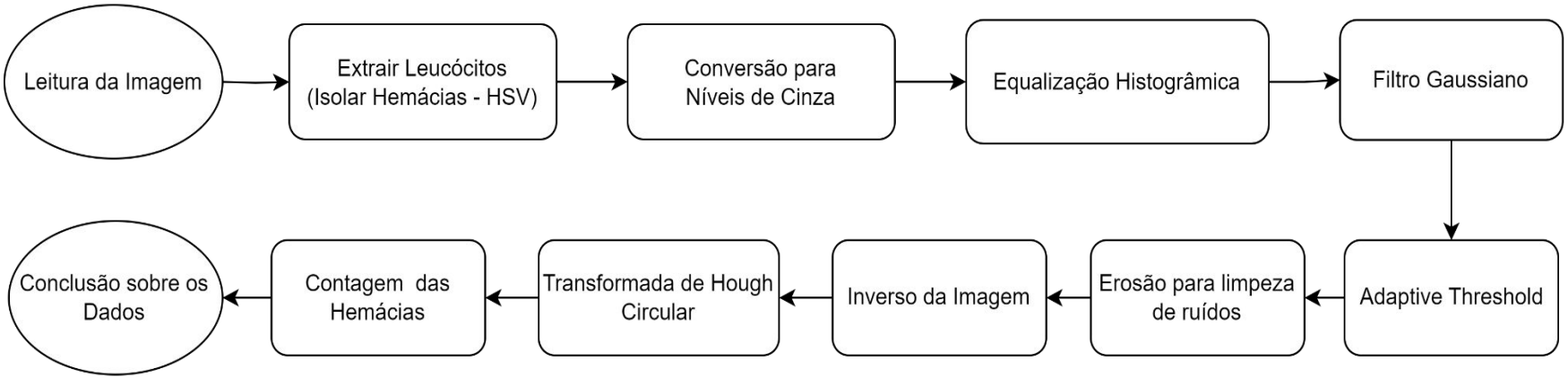
https://github.com/MahmudulAlam/Complete-Blood-Cell-Count-Dataset

O *dataset* disponibiliza *360 imagens de hemogramas*.

Entretanto, ao longo do desenvolvimento do projeto, escolheu-se um **espaço amostral** de ***6 imagens*** do *dataset* e inferiu-se conclusões em relação à **precisão das técnicas** utilizadas para o processamento dessas imagens.

Desenvolvimento

O seguinte **fluxograma** resume o processo de desenvolvimento do processamento das imagens dos hemogramas para a detecção das hemácias



Extração Leucócitos (Isolar Hemácias - HSV)

Seguindo a sugestão do professor, exploramos os **canais HSV** e nos concentramos especialmente no **canal H**, que define **a cor principal de um pixel**. A ideia central aqui é encontrar o "*intervalo HSV"* que corresponde aos Leucócitos presentes nas imagens e, a partir disso, formar uma máscara que será usada para excluir os Leucócitos. Para isso alguns passos foram feitos:

- Passo 1: Encontrar os valores Máximos de HSV;
- Passo 2: De forma manual, encontrar o intervalo de H1 e H2 que pertence ao leucócito.
- Passo 3: Criar uma máscara/imagem baseada no hsv (arredondado para a dezena de cima mais próxima) da imagem a somente com valores do intervalo do passo 2.
- Passo 4: Fazer a Limpeza da imagem aplicando o fechamento e depois a abertura
- Passo 5: Gerar o conjunto complementar da máscara filtrada (Obtendo tudo menos os intervalo HSV que contém os leucócitos)
- Passo 6: Fazer a interseção da máscara com a imagem Gerando uma nova imagem somente com hemácias

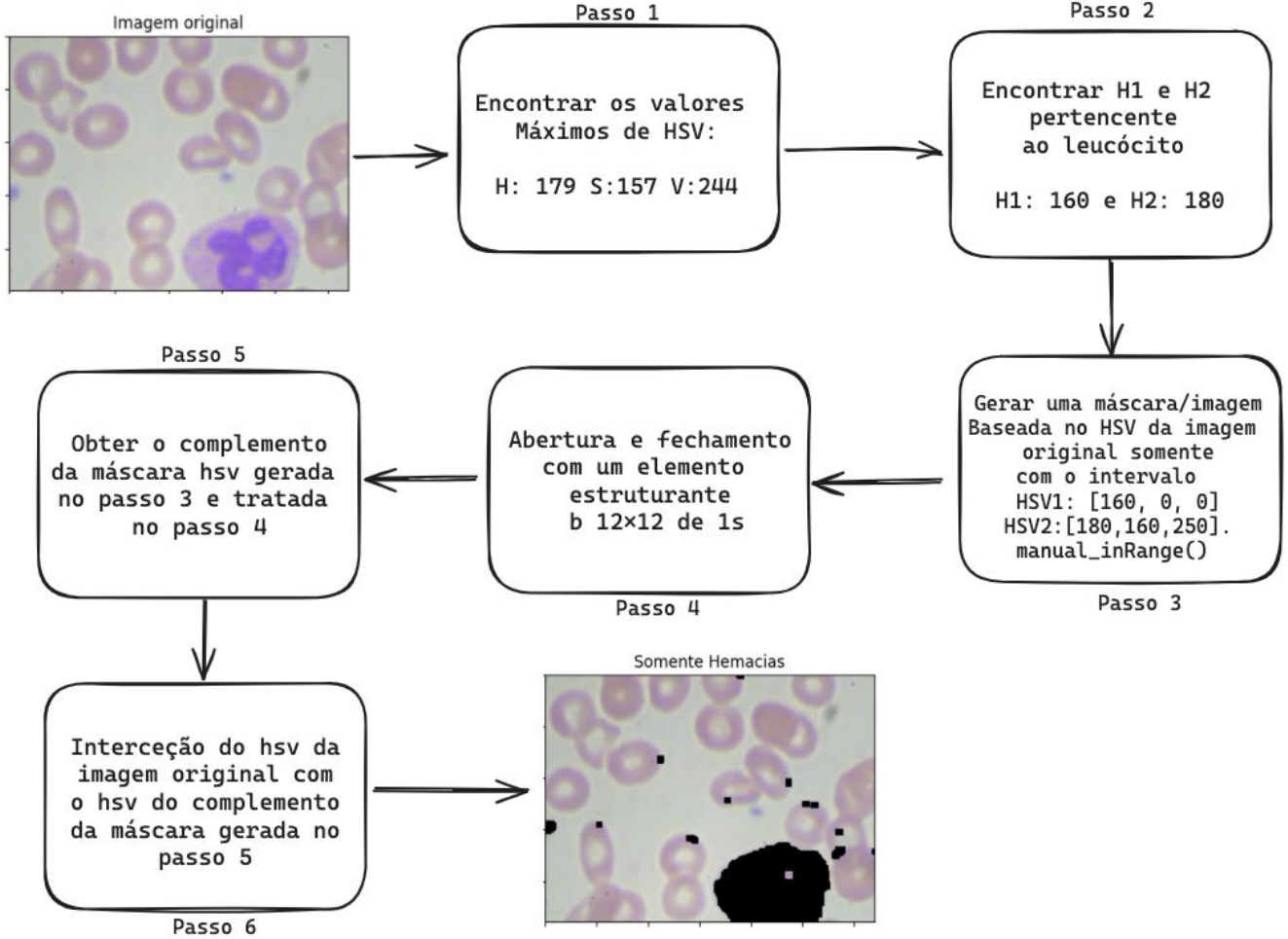


Imagem original
Passo 1
Encontrar os valores Máximos de HSV:
H: 179 S:157 V:244
Passo 2
Encontrar H1 e H2 pertencente ao leucócito
H1: 160 e H2: 180
Gerar uma máscara/imagem Baseada no HSV da imagem original somente com o intervalo HSV1: [160, 0, 0] HSV2:[180,160,250]. manual_inRange()
Passo 3
Abertura e fechamento com um elemento estruturante b 12×12 de 1s
Passo 4
Passo 5
Obter o complemento da máscara hsv gerada no passo 3 e tratada no passo 4
Interceção do hsv da imagem original com o hsv do complemento da máscara gerada no passo 5
Passo 6
Somente Hemacias

Extração Leucócitos (Isolar Hemácias - HSV)

As definições das funções:

- A função ***manual_inRange()*** gera uma máscara somente com os leucócitos
- A função ***manual_bitwise_and()*** é responsável por aplicar a interseção da imagem de entrada com o complemento da máscara de leucócitos (Somente hemácias) se o parâmetro negacao=True

```python
def manual_inRange(hsv, lower, upper):

    h, s, v = hsv[:,:,0], hsv[:,:,1], hsv[:,:,2]

    h_mask = np.logical_and(h >= lower[0], h <= upper[0])
    s_mask = np.logical_and(s >= lower[1], s <= upper[1])
    v_mask = np.logical_and(v >= lower[2], v <= upper[2])

    return (h_mask & s_mask & v_mask).astype(np.uint8) * 255
```

```python
def manual_bitwise_and(img1, mask, negacao=False):
    if  img1.shape[:2] != mask.shape[:2]:
        raise ValueError("O tamanho da imagem e da mascara deve ser iguais.")

    h, w, _ = img1.shape

    result_img = np.zeros((h, w, _), dtype=np.uint8)

    for i in range(h):
        for j in range(w):
            # pega o complementar da imagem ou não a partir do complementar da mascara
            mask_tratada = ~mask[i, j] if negacao else mask[i, j]
            result_img[i, j] = img1[i, j] & mask_tratada # Aplica o bitwise

    return result_img
```

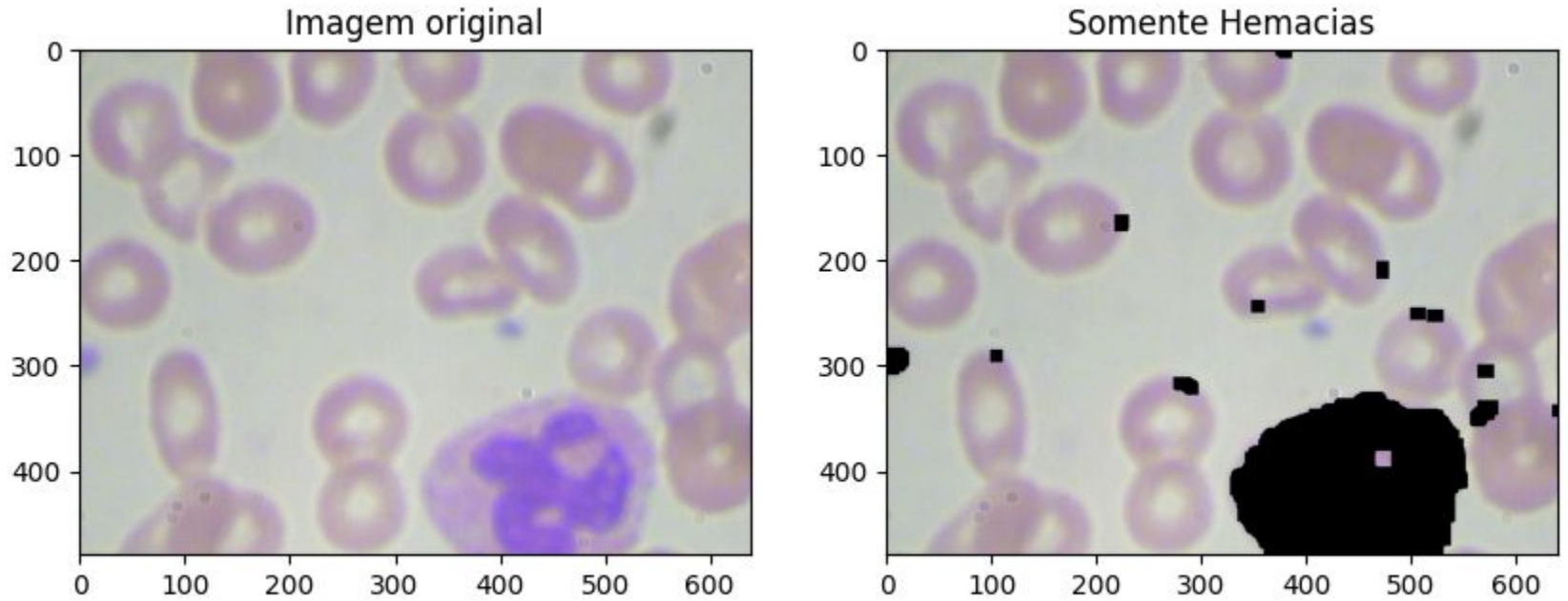
Extração Leucócitos (Isolar Hemácias - HSV)

Por fim, esta é a função principal que sintetiza a exclusão dos Glóbulos Brancos (Leucócitos):

```python
def RemoveGlobulosBrancos(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Intervalo que se encontram os globulos brancos
    lower = np.array([160, 0, 0])
    upper = np.array([180, 161, 247])

    mask = manual_inRange(hsv, lower, upper) # obtém a mascara com o intervalo

    # Faz a limpeza de ruidos que ficam na imagem. Aplicando o fechamento e após a abertura
    b = np.ones((12,12),np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, b)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, b)

    return manual_bitwise_and(img, mask=mask, negacao=True)
```

Extração Leucócitos (Isolar Hemácias - HSV)

Aplicando essa rotina em uma imagem, esse é o resultado obtido:



Pré-processamento: Conversão para Níveis de Cinza, Equalização Histogrâmica e Filtro Gaussiano

Com a imagem *filtrada*, faremos agora o seu pré-processamento:

1. Conversão para **níveis de cinza**.

2. **Equalização histogrâmica**, que serve para melhorar o contraste das células em relação ao seu background, ajudando mais tarde na *binarização da imagem*.

3. Aplicação do **Filtro Gaussiano**, que reduz o *ruído* e ajuda a detectar as *bordas* das células, auxiliando também no *Threshold (Linearização)*.

Pré-processamento: Conversão para Níveis de Cinza, Equalização Histogrâmica e Filtro Gaussiano

Por fim, esta é a função que sintetiza essa etapa de pré-processamento da imagem:

```python
def preprocessamento(hemacias):

    # Conversão para cinza
    gray_redCells = cv2.cvtColor(hemacias, cv2.COLOR_BGR2GRAY)

    # Equaliza a imagem
    eq_img = cv2.equalizeHist(gray_redCells)

    # Aplica o filtro gaussiano 5x5 pra remover ruidos indesejados
    k_mask_gaussian = 5
    mask_gaussian = (k_mask_gaussian, k_mask_gaussian)
    gaussian_img = cv2.GaussianBlur(eq_img, mask_gaussian, 0)

    return gray_redCells, eq_img, gaussian_img
```

Adaptive Threshold (Limiarização Adaptativa)

Como comentado anteriormente, decidimos adotar uma abordagem de limiarização diferente da abordada em aula a fim de chegar em melhores resultados.

A **Limiarização Adaptativa (Adaptive Threshold)** diferente da *Limiarização Simples (Simples Threshold)*, a discutida em aula, não utiliza um valor *global* como limiar para binarizar a imagem. Mas sim, determina um valor de limiar *local* para cada pixel da imagem baseado em uma pequena região ao seu redor.

Aliás, essa abordagem é mais **eficiente** para imagens que têm condições de luminosidade diferentes em diferentes áreas.

Em nossos testes, essa técnica trouxe melhores resultados, tendo em vista que as imagens do *dataset* possuem uma variação de luminosidade.

Adaptive Threshold (Limiarização Adaptativa)

Agora, falta definir qual abordagem usaremos para determinar o limiar local de um pixel. Comumente, calcula-se os limiares a partir da **média** da vizinhança ou do **desvio padrão** dos valores de intensidade em regiões específicas da imagem.

No nosso projeto, testamos exaustivamente tanto a média quanto o desvio padrão e com base nos testes, optamos pela média pois foi ela que nos deu o melhor resultado.

Além desse parâmetro inicial, é necessário definir outros parâmetros:

- *block_size*: tamanho da vizinhança do pixel.
- *C*: constante que é subtraída do resultado do cálculo do limiar.
- *max_level*: valor que é adotado quando a condição é verdadeira.

Adaptive Threshold (Limiarização Adaptativa)

Considerando um exemplo trivial de apenas uma iteração de Adaptive Threshold usando a média para uma matriz aleatória 5x5 para *vizinhança* de 3 pixels e C = 7.

Matriz 5x5 qualquer

230	26	16	221	98
201	125	250	227	194
183	102	216	209	94
63	95	20	202	16
250	46	122	118	229

Calculando o limiar para o pixel (2, 2)

230	26	16	221	98
201	125	250	227	194
183	102	216	209	94
63	95	20	202	16
250	46	122	118	229

Calculando a média da vizinhança

230	26	16	221	98
201	125	250	227	194
183	102	216	209	94
63	95	20	202	16
250	46	122	118	229

Resultado da iteração

		0		

Para o pixel central, temos como média o valor de 161, que subtraído por 7 resulta em 154.

O valor do pixel (216) < Limiar (154), ou seja, o pixel é levado para 0.

Adaptive Threshold (Limiarização Adaptativa)

Com isso, foi feita uma implementação própria desse algoritmo.

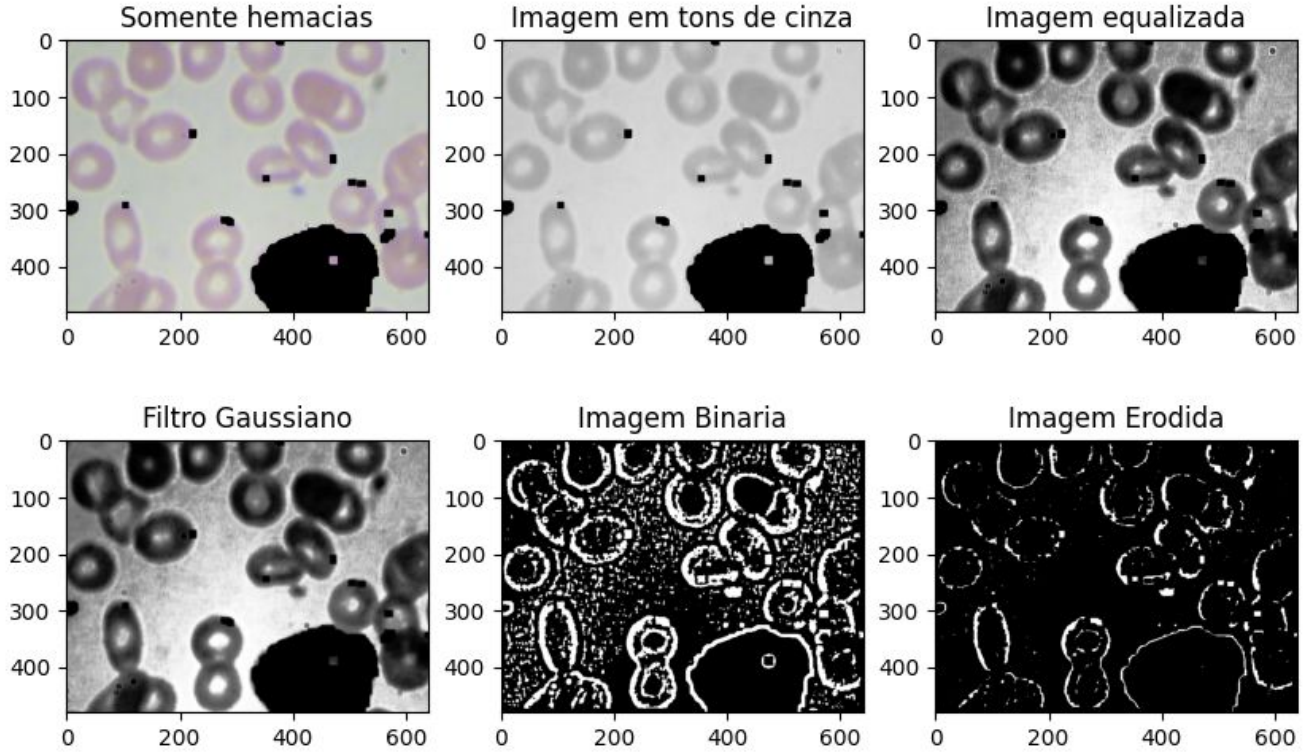
```python
def meu_adaptiveThreshold(img, max_level, metodo, type_threshold, block_size, C):

    tamanho_borda = block_size // 2

    # Adiciona borda à imagem de entrada
    borda_img = np.pad(img, tamanho_borda, mode='constant')

    img_saida = np.zeros_like(img)

    j, i = img.shape
    for y in range(tamanho_borda, j + tamanho_borda):
        for x in range(tamanho_borda, i + tamanho_borda):
            regiao_sele = borda_img[y - tamanho_borda:y + tamanho_borda + 1, x - tamanho_borda:x + tamanho_borda + 1]

            if metodo == 0:
                threshold_value = np.mean(regiao_sele) - C # Aplica o threshold da media
            else:
                 # aplica o threshold com desvio padrao
                threshold_value = np.mean(regiao_sele) - C * np.std(regiao_sele) / block_size

            # Essa região aplica a binarização se satisfeita as condições da região
            if type_threshold == 0:
                img_saida[y - tamanho_borda, x - tamanho_borda] = max_level if img[y - tamanho_borda, x - tamanho_borda] > threshold_value else 0

            else:
                img_saida[y - tamanho_borda, x - tamanho_borda] = 0 if img[y - tamanho_borda, x - tamanho_borda] > threshold_value else max_level

    return img_saida
```

Adaptive Threshold (Limiarização Adaptativa)

A fim de sintetizar esse processo de binarização da imagem, foi feita esta função:

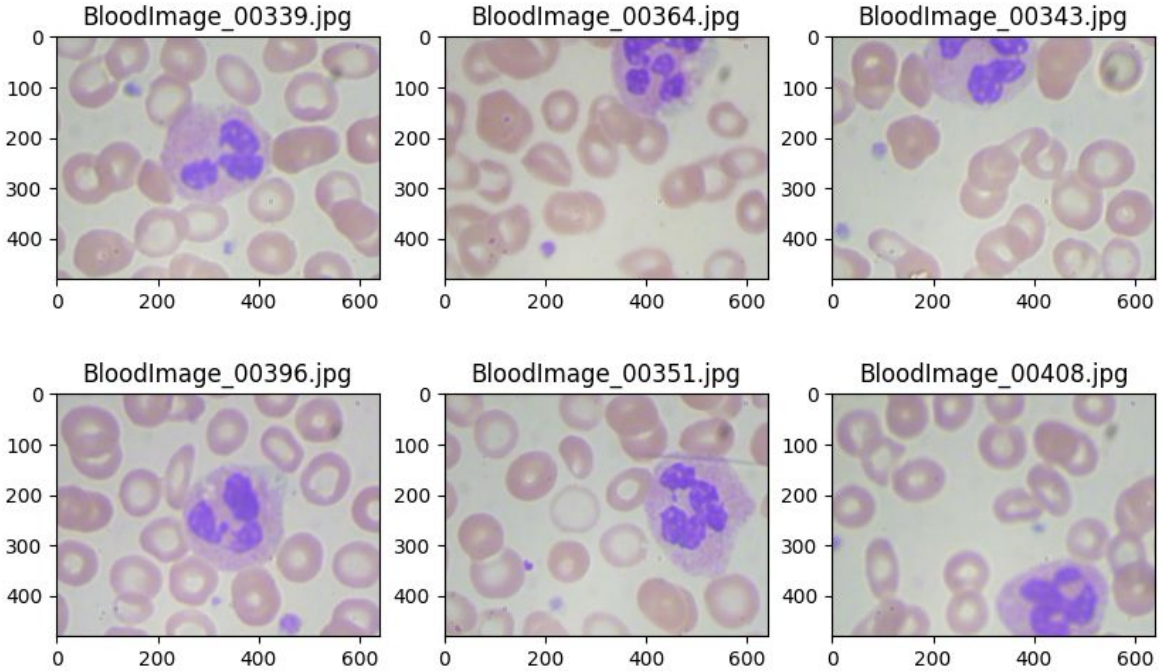
```python
def binariza_imagem(img_processada):
    '''
    Binariza a imagem  de forma adaptativa com o uso do método obtivemos os
    melhores resultados para as imagens selecionadas ajustamos para
    '''

    img_binary = meu_adaptiveThreshold(img_processada, 255, 0, 1, 15, 7)

    # Aplicacao da erosao para fazer uma limpeza nos ruídos
    k = 5
    i = 5
    kernel = np.ones((k,k),np.uint8)
    img_erodida = cv2.erode(img_binary, kernel, i)

    inverso = 255-img_erodida

    return img_binary, img_erodida, inverso
```

Um teste de todos os passos apresentados até agora…



Metodologia da detecção de círculos (Imagens Selecionadas)

Foram selecionadas seis imagens que tivessem diferentes disposição de elementos e aparentavam cores diferentes umas das outras.



Metodologia da detecção de círculos (Critérios da detecção)

Após isso foram feitos incansáveis ajustes manuais para obter o melhor resultado entre todas as imagens e estabelecemos algumas condições para isso:

1. Ter a maior quantidade de verdadeiro positivos possíveis
2. Ter entre 0 e 4 falsos positivos (FP)
3. Ter mais de 55% do Verdadeiros positivos (VP)
4. Ter menos de 45% de falsos negativos (FN)

Metodologia da detecção de círculos.

A função ao lado aplica a Transformada Circular de Hough na imagem binarizada.

A Transformada gera uma grade de acumuladores que é usada para contar e desenhar os círculos correspondentes às hemácias na imagem.

```python
def identifica_hemacias(img, binary_image):

    celulas = img.copy()

    circles = cv2.HoughCircles(binary_image,cv2.HOUGH_GRADIENT,1,68,
                               param1=33,param2=12,minRadius=30,maxRadius=58)

    hemacias_count = 0

    circles = np.uint32(np.around(circles))
    for i in circles[0,:]:
        cv2.circle(celulas,(i[0],i[1]),i[2],(0,255,0),2)
        cv2.circle(celulas,(i[0],i[1]),2,(0,0,255),3)

        hemacias_count +=1


    return celulas, hemacias_count
```

Metodologia da detecção de círculos (Seleção de Parâmetros)

```
circles = cv2.HoughCircles(binary_image,cv2.HOUGH_GRADIENT,1,68,
                           param1=33,param2=11,minRadius=30,maxRadius=58)
```

Com base em vários testes manuais aplicamos a função ***HoughCircles*** com alguns parâmetros específicos que geraram resultados satisfatórios.

A seguir uma breve explicação sobre os parâmetros da função:

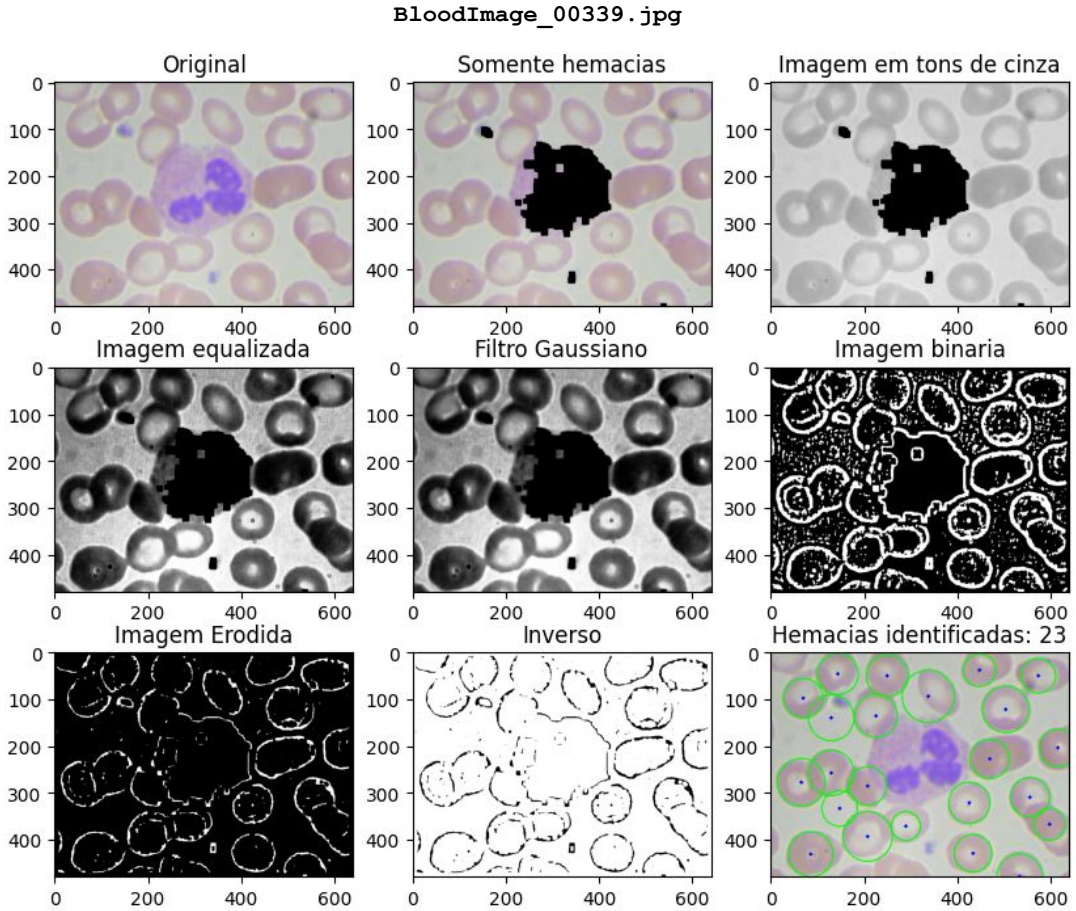
- O **primeiro parâmetro** é a imagem binarizada.
- O **segundo parâmetro** indica que é usada a técnica do gradiente para detectar **bordas** na imagem. Ela explora as mudanças bruscas de intensidade dos pixels da imagem que frequentemente correspondem às bordas de objetos na região.
- O **terceiro parâmetro** com 1 indica que a **resolução** da imagem de entrada é **mantida** na aplicação da transformada de Hough.
- O **quarto parâmetro** é a distância mínima entre os centros dos círculos detectados. Se a distância entre os centros for menor que esse valor, apenas o de maior confiança (que teve mais match com elementos) que será adicionado. Durante alguns testes no intervalo de 50 a 70, vimos que o melhor valor para o nosso caso está entre 65 e 70, optamos por deixar em 68.

Metodologia da detecção de círculos (Seleção de Parâmetros)

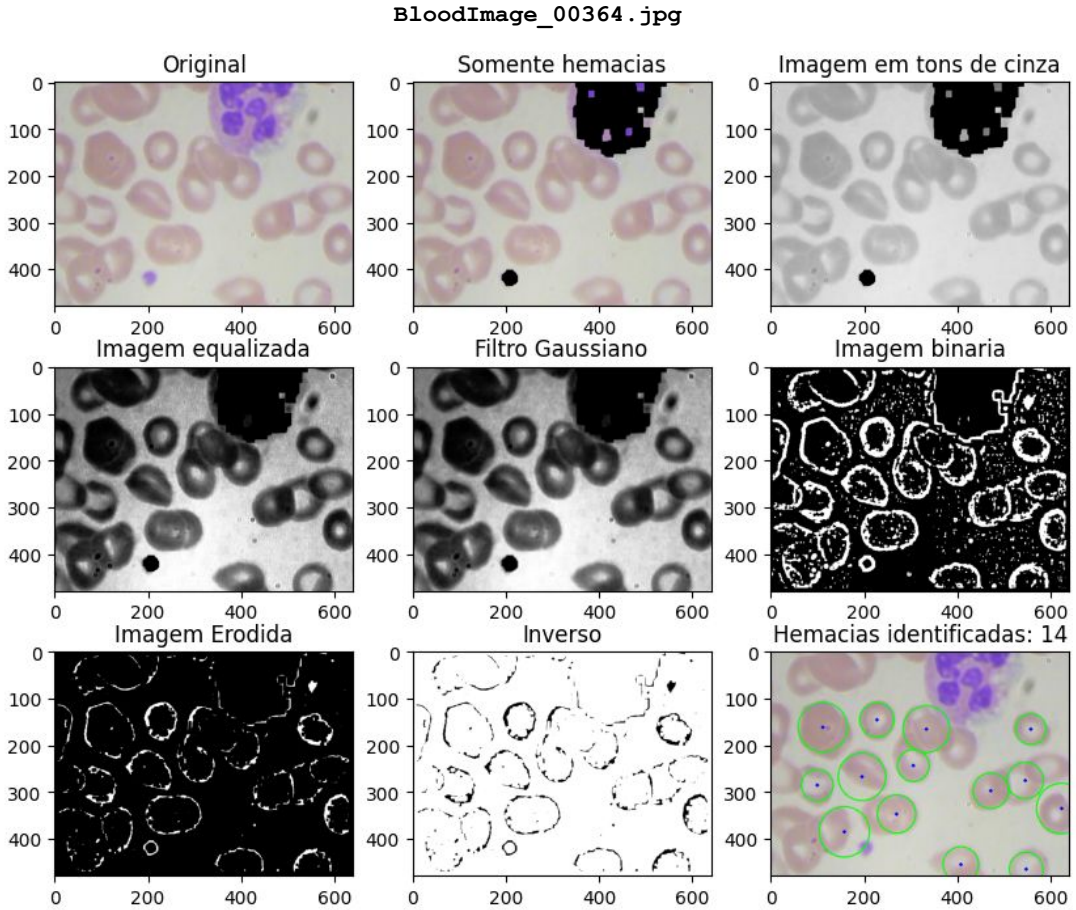
```
circles = cv2.HoughCircles(binary_image,cv2.HOUGH_GRADIENT,1,68,
                           param1=33,param2=11,minRadius=30,maxRadius=58)
```

- O **quinto parâmetro** (*param1 = 50*) é usado para definir a **sensibilidade do detecto**r de bordas interno do *HOUGH_GRADIENT*. Valores menores de param1 resultarão em mais círculos sendo detectados (possivelmente com mais *falsos positivos*). Durante alguns testes no intervalo de 20 a 70, 33 foi um bom número para nós.
- O **sexto parâmetro** (*param2 = 11*) é usado para definir o **limite mínimo para a detecção de bordas**. Valores menores de param2 resultarão em mais círculos sendo detectados, mas também em mais falsos positivos. Geralmente o param2 é igual a param1 dividido por 3, e esse foi o parâmetro usado. Testamos ele de 5 a 33 e esse foi o melhor valor.
- O **sétimo parâmetro** é *minRadius = 30.* Esse é o raio mínimo do círculo a ser detectado, ou seja, qualquer círculo com um raio menor do que este valor não será detectado. Testamos minRadius de 20 a 40, e 30 foi o melhor equilíbrio entre identificar poucos falsos positivos, falsos negativos e uma quantidade alta de Verdadeiros positivos.
- O **oitavo parâmetro** é *maxRadius = 58* que quer dizer que qualquer círculo com um raio maior do que este valor não será detectado. Durante diversos testes com valores entre 45 e 70, encontramos o valor 58 como o mais satisfatório para os nossos objetivos.

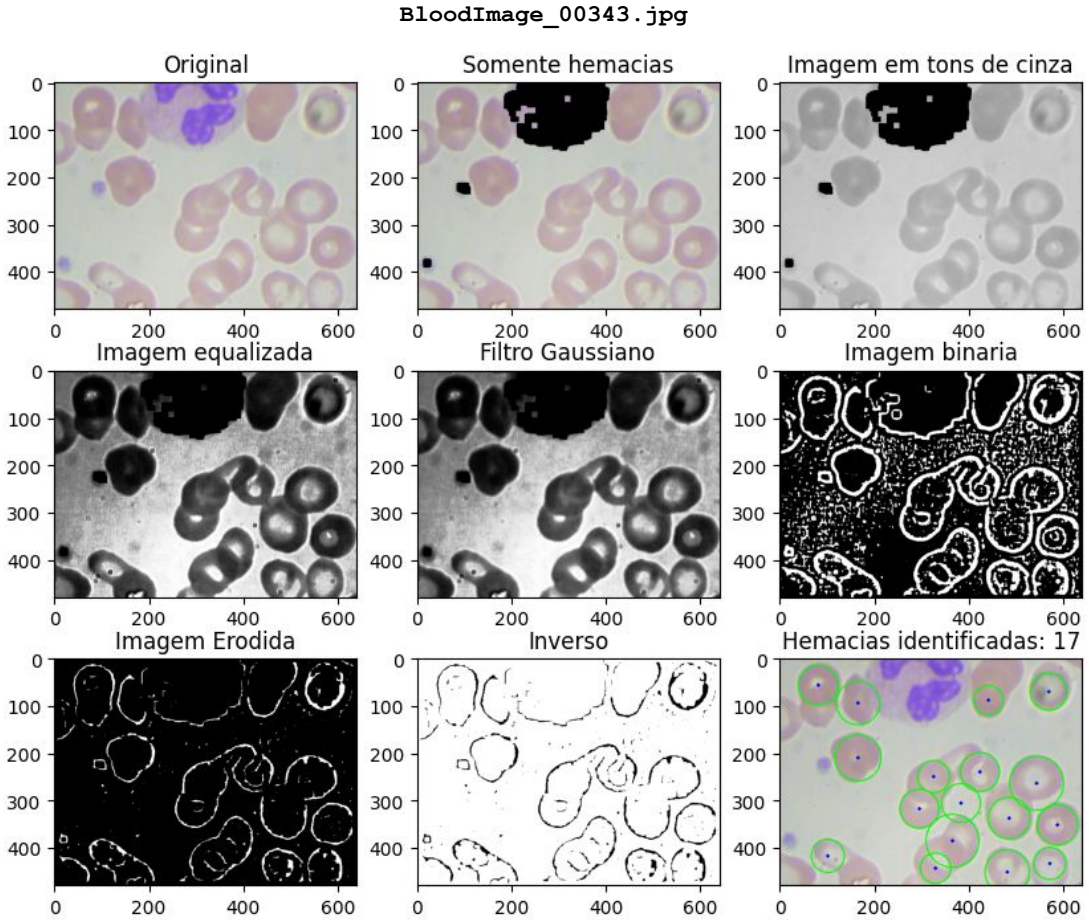
Metodologia da detecção de círculos (Aplicação)



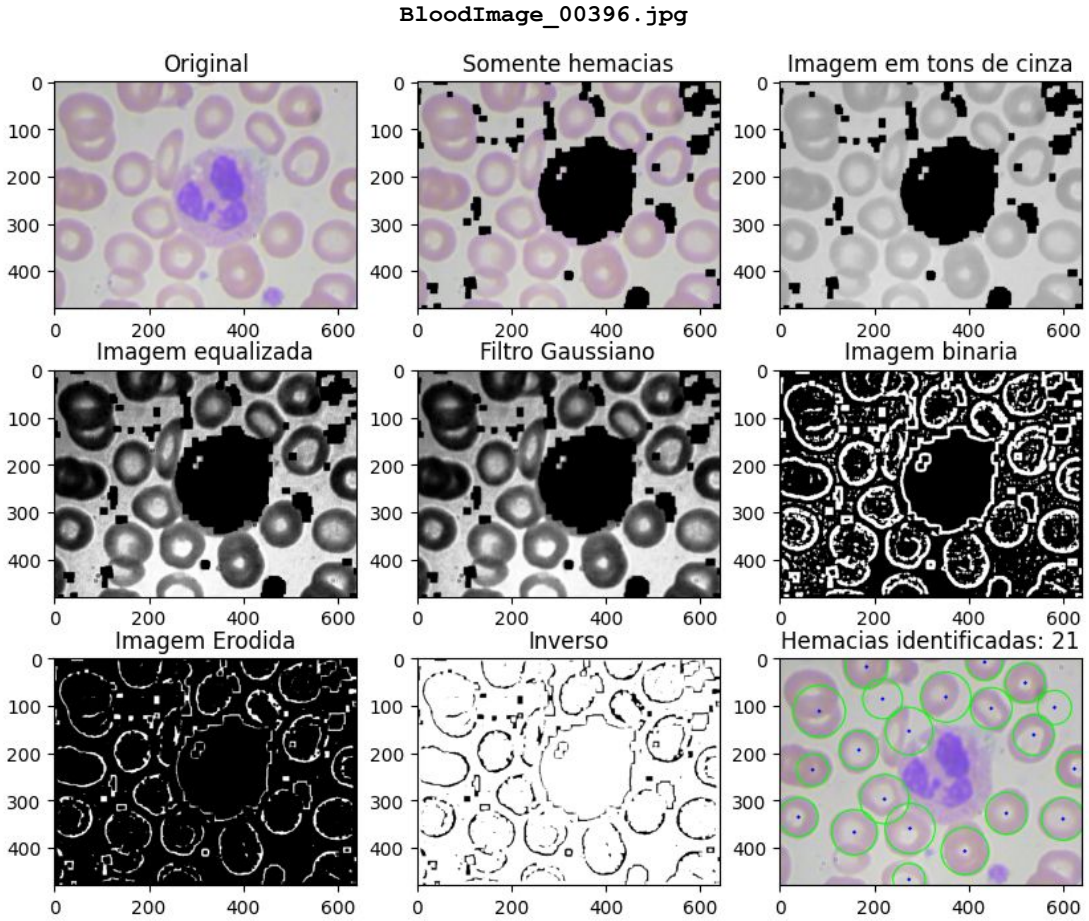
Metodologia da detecção de círculos (Aplicação)



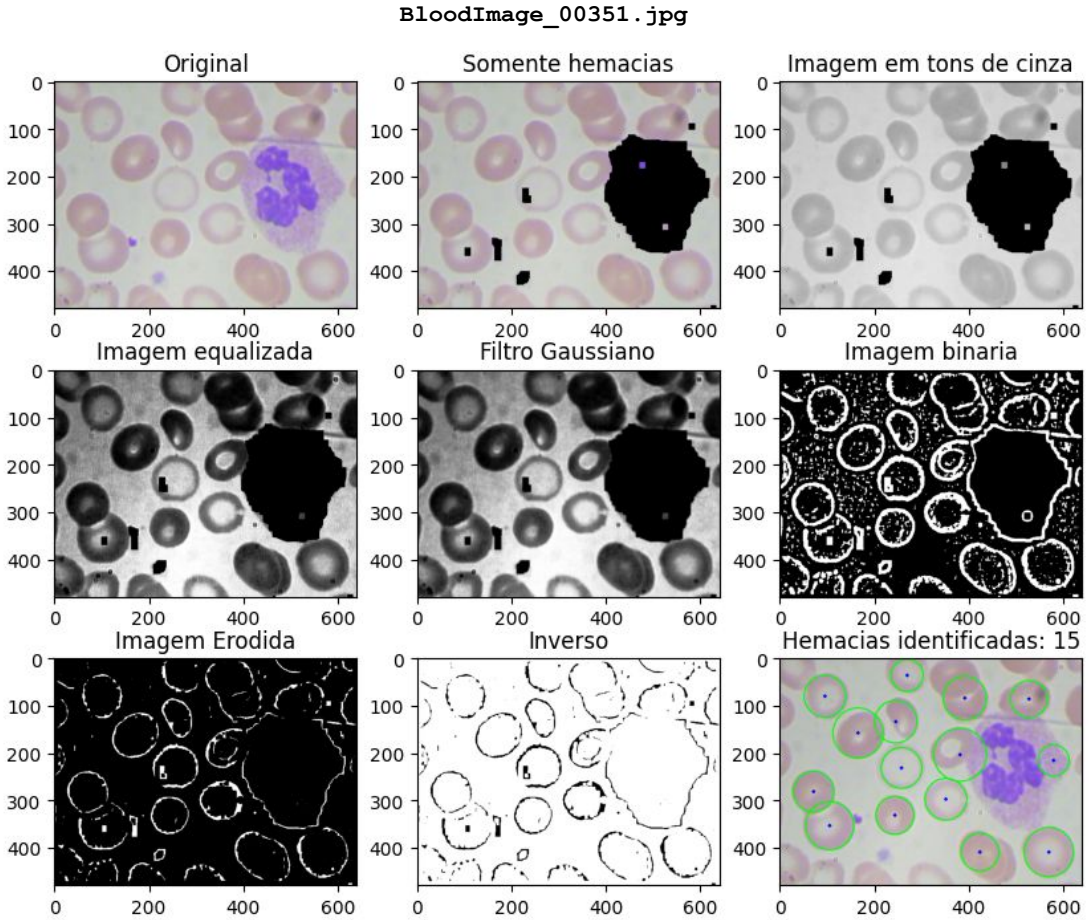
Metodologia da detecção de círculos (Aplicação)



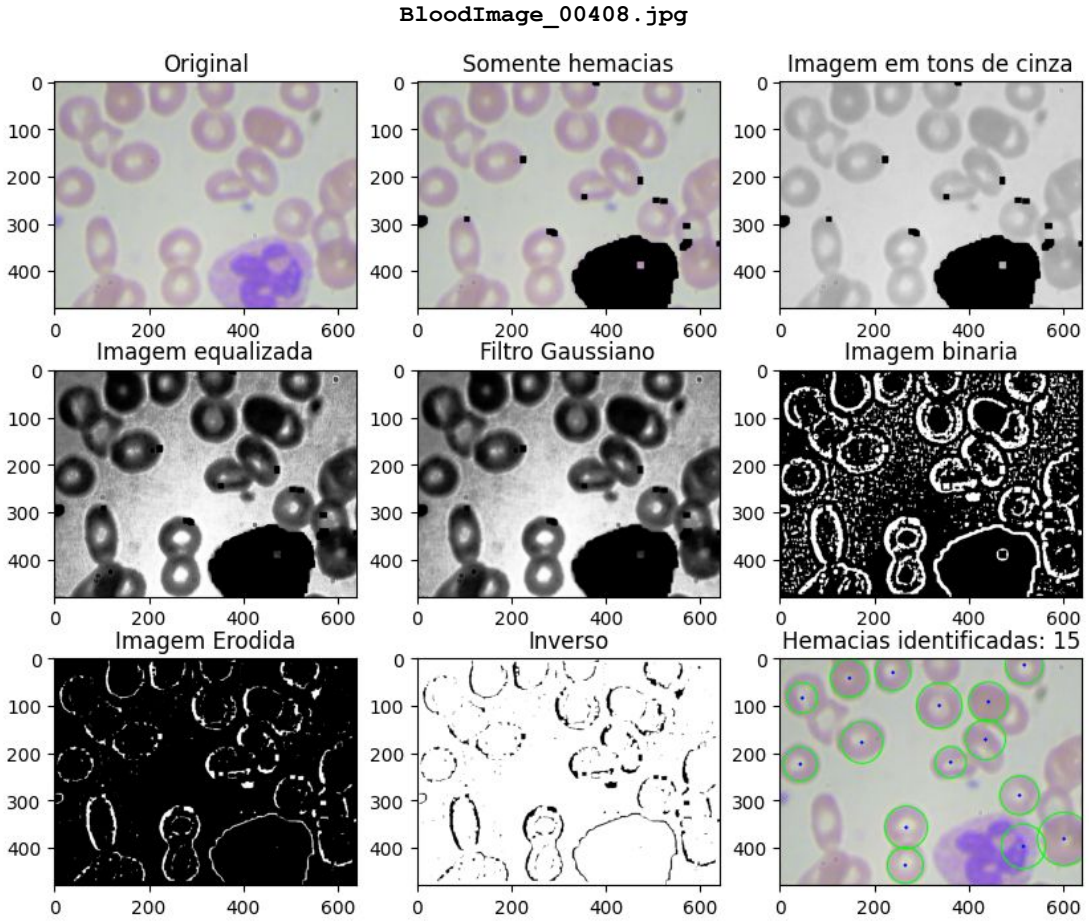
Metodologia da detecção de círculos (Aplicação)



Metodologia da detecção de círculos (Aplicação)



Metodologia da detecção de círculos (Aplicação)



Análise de Dados e Conclusão

Antes de começar, algumas observações precisam ser feitas:

- Caso haja a diminuição da tolerância de FN, a quantidade de FP aumenta, trazendo a possível detecção em glóbulos brancos.

- Na **contagem manual**, evitamos contar hemácias que não apareciam pelo menos **20%** na imagem, pois não há uma certeza de que sejam hemácias.

- Optamos pela contagem somente das hemácias evitando leucócitos, claro que tiveram alguns resultados que foi contabilizado um círculo em alguma cavidade de algum leucócito (falso positivo).

Análise de Dados e Conclusão

Algumas métricas para medir a precisão:

- **taxaFN(FN, VP)** : Calcula a taxa de *Falsos Negativos (FN)* em porcentagem. *(FN / (FN + VP))*

- **taxaVP(VP, FN)**: Calcula a taxa de *Verdadeiros Positivos (VP)* em porcentagem. ou recall *VP / (VP + FN)*

- **precisao(VP, FP)**: Calcula a *precisão* do modelo, basicamente é o quanto o modelo acerta o que é hemácia de fato, quanto mais próximo de 1 melhor. *VP/ (VP + FP)*

- **F1Score(precisao, recall)**: média harmônica da precisão e da revocação. Ela fornece uma única medida que combina precisão e recall em um único número. O F1 Score varia de 0 a 1, onde 1 indica um modelo perfeito e 0 indica o pior desempenho possível. *2 * ((precisao * taxaVP) / (precisao + taxaVP))*

Análise de Dados e Conclusão

A seguir, a definição das funções para cálculo das métricas:

```python
# Calculos das taxas
def taxaFN(FN, VP):
  return "{:.2f}%".format((FN / (FN + VP)) * 100)

def taxaVP(VP, FN):
  c = VP / (VP + FN)
  return "{:.2f}%".format(c * 100), c

def precisao(VP, FP):
  c = VP/ (VP + FP)
  return "{:.2f}".format(c), c


def F1Score(precisao, recall):
  return 2 * ((precisao * recall) / (precisao + recall))
```

Análise de Dados e Conclusão

Após a execução dos cálculos das taxas, o resultado obtido foi resumido em uma tabela, que é apresentada adiante:

Tabela 1 - Detalhamento das Métricas

Imagem	FN	FP	VP	Qtd existente	Taxa VP/Recall	Taxa FN	Precisão	F1 Score
BloodImage_00339.jpg	7	2	21	28	75,00%	25,00%	0,91	0,823529
BloodImage_00364.jpg	9	0	13	22	59,09%	40,91%	1,00	0,742857
BloodImage_00343.jpg	6	1	16	22	72,73%	27,27%	0,94	0,820513
BloodImage_00396.jpg	8	2	19	27	70,37%	29,63%	0,90	0,791667
BloodImage_00351.jpg	9	2	15	24	62,50%	37,50%	0,88	0,731707
BloodImage_00408.jpg	9	1	14	23	62,87%	39,13%	0,93	0,736842

Análise de Dados e Conclusão

Tabela 2 - Médias da Tabela 1

Média de cada coluna	
FN	8,000000
FP	1,333333
VP	16,333333
Qtd existente	24,333333
Taxa VP/Recall	0,667600
Taxa FN	0,332400
Precisão	0,926667
F1 Score	0,774519

Na Tabela 2 está a ***média*** de cada coluna da Tabela 1. Vamos nos atentar a dois dados principalmente:

- ***Precisão de 93%***, que indica que podemos confiar 0.93 que o que nossa solução está identificado são realmente hemácias

- ***F1 Score de 77%***, sugere que o modelo de detecção está alcançando um bom equilíbrio entre precisão e recall na identificação de hemácias nas imagens analisadas.

Análise de Dados e Conclusão

Observações Finais:

- Obtivemos resultados **satisfatórios**, com uma margem de desempenho médio de **70% F1 Score** e uma **precisão de 90%** com as técnicas aprendidas na matéria de Processamento de Imagens.

- Outro ponto é na questão da **limitação** dos métodos empregados. Pois, mesmo trazendo bons resultados, uma abordagem com Aprendizado de Máquina provavelmente traria resultados mais expressivos.

Referências

- https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d (Método HoughCircles)

- Gonzalez, R. and Woods, R. Digital Image Processing. Prentice Hall, 3nd. Edition, 2010.

- Pedrini, H. and Schwartz, W.R. Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. Thomson, 2007.

- https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html (Método HSV)

- Rodrigues, Vitor Borba. "Métricas de Avaliação: Acurácia, Precisão, Recall - Quais as Diferenças?" Medium, 2019, https://vitorborbarodrigues.medium.com/m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-recall-quais-as-diferen%C3%A7as-c8f05e0a513c.