UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
LABORATÓRIO DE PESQUISAS APLICADAS À VISÃO E INTELIGÊNCIA COMPUTACIONAL (PAVIC)

TUTORIAL - PROCESSAMENTO DE IMAGENS

Picos
2018

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

Acc	**Acurácia**
CAD	**Computed Aided Diagnosis**
Esp	**Especificidade**
FN	**Falso Negativo**
FP	**Falso Positivo**
mACC	**Média da Acurácia**
mESP	**Média da Especificidade**
mROC	**Média da Área sob a curva ROC**
mSEN	**Média da Sensibilidade**
MVS	**Máquina de Vetor de Suporte**
PDI	**Processamento Digital de Imagens**
Sen	**Sensibilidade**
VN	**Verdadeiro Negativo**
VP	**Verdadeiro Positivo**
ROI	**Region of Interest**
KERNEL	**Núcleo da imagem**

1. INTRODUÇÃO

O foco deste manual é fornecer a base inicial de conhecimento para os iniciantes acerca de visão computacional e processamento digital de imagens, com o intuito de que possam compreender cada uma das etapas de processamento, bem como as técnicas e características envolvidas em cada uma delas, para, assim, obterem resultados eficientes e eficazes dentro do laboratório de Pesquisas Aplicadas à Visão e Inteligência Computacional (PAVIC). O manual contará com as principais etapas do processamento de imagens, como técnicas, ferramentas para auxiliar o aluno-pesquisador, exemplos de códigos e informações que servirão de base para pesquisas com aprofundamento específico nas métricas escolhidas por cada aluno-pesquisador integrante.

O processamento digital de imagens atualmente possui aplicações em diversas áreas, abrangendo desde geologia, que estuda, por exemplo, a composição da superfície, agricultura, engenharia florestal, cartografia e meteorologia. Crescentes áreas de pesquisas são encontradas na medicina, biologia, astronomia, física, segurança, monitoramento, etc.

2. PROCESSAMENTO DE IMAGENS

O Processamento Digital de Imagens (PDI) não é uma tarefa simples, na realidade envolve um conjunto de tarefas interconectadas. Tudo se inicia com a captura de uma imagem, a qual, normalmente, corresponde à iluminação que é refletida na superfície dos objetos, realizada através e um sistema de aquisição. Após a captura por um processo de digitalização, uma imagem precisa ser representada de forma apropriada para tratamento computacional. Imagens podem ser representadas em duas ou mais dimensões. O primeiro passo efetivo de processamento é comumente conhecido como pré-processamento, o qual envolve passos como a filtragem de ruídos introduzidos pelos sensores e a correção de distorções geométricas causadas pelo sensor.

Uma cadeia maior de processos é necessária para a análise e identificação de objetos. Primeiramente, características ou atributos das imagens precisam ser extraídos, tais como as bordas, texturas e vizinhanças. Outra característica importante é o movimento. Em seguida, objetos precisam ser separados do plano de fundo (background), o que significa que é necessário identificar, através de um processo de segmentação, características constantes e descontinuidades. Esta tarefa pode ser simples, se os objetos são facilmente destacados da imagem de fundo, mas normalmente este não é o caso, sendo necessárias técnicas mais sofisticadas como regularização e modelagem. Essas técnicas usam várias estratégias de otimização para minimizar o desvio entre os dados de imagem e um modelo que incorpora conhecimento sobre os objetos da imagem. Essa mesma abordagem matemática pode ser utilizada para outras tarefas que envolvem restauração e reconstrução. A partir da forma geométrica dos objetos, resultante da segmentação, pode-se utilizar operadores morfológicos para analisar e modificar essa forma bem como extrair

informações adicionais do objeto, as quais podem ser úteis na sua classificação. A classificação é considerada como uma das tarefas de mais alto nível e tem como objetivo reconhecer, verificar ou inferir a identidade dos objetos a partir das características e representações obtidas pelas etapas anteriores do processamento. Como último comentário, deve-se observar que, para problemas mais difíceis, são necessários mecanismos de retro-alimentação (feedback) entre as tarefas de modo a ajustar parâmetros como aquisição, iluminação, ponto de observação, para que a classificação se torne possível. Esse tipo de abordagem também é conhecido como visão ativa. Em um cenário de agentes inteligentes, fala-se de ciclos de ação-percepção.

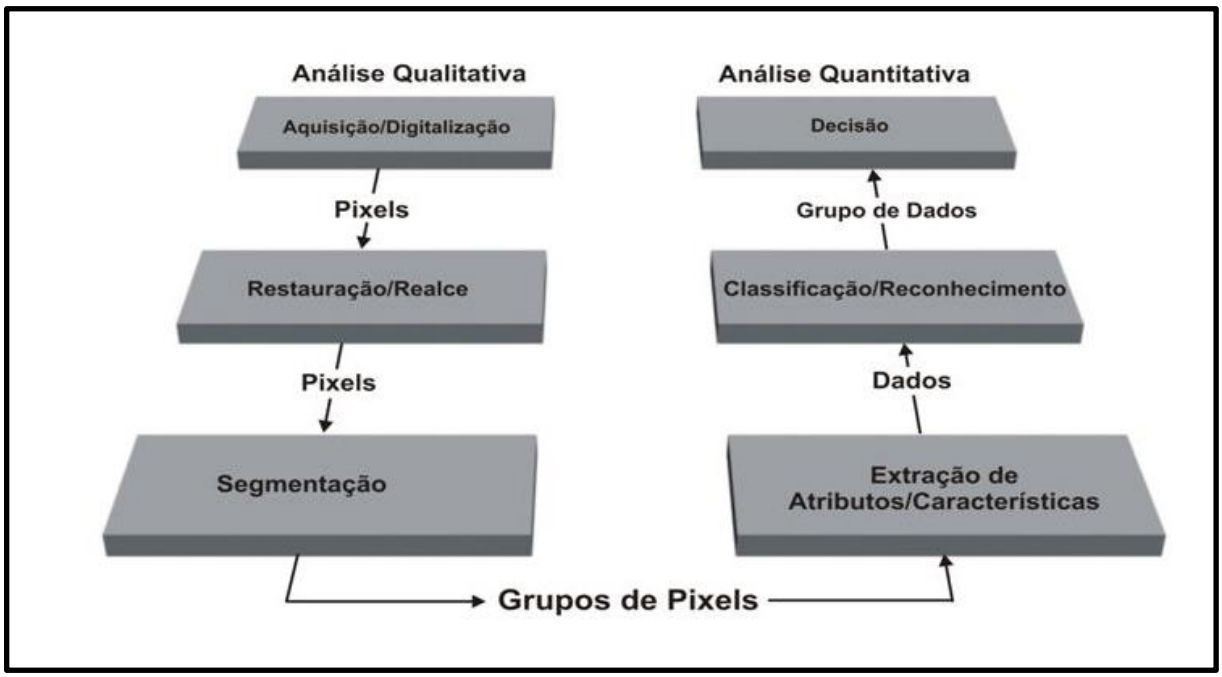

Fig. 1 - Etapas de um sistema de vc genérico

A área de Processamento de Imagens incorpora fundamentos de várias ciências, como Física, Computação, Matemática. Conceitos como Óptica, Física do Estado Sólido, Projeto de Circuitos, Teoria dos Grafos, Álgebra, Estatística, dentre outros, são comumente requeridos no projeto de um sistema de processamento de imagens. Existe também uma intersecção forte entre PDI e outras disciplinas como Redes Neurais, Inteligência Artificial, Percepção Visual, Ciência Cognitiva. Há igualmente um número de disciplinas as quais, por razões históricas, se desenvolveram de forma parcialmente independente do PDI, como Fotogrametria, Sensoriamento Remoto usando imagens aéreas e de satélite, Astronomia e Imageamento Médico.

Adiante, serão apresentadas cada uma das etapas do processamento digital de imagens, seguidas de exemplos sobre as operações e técnicas envolvidas.

3. MANIPULAÇÃO DE IMAGENS

No presente tópico e subtópicos serão apresentados definições básicas para o entendimento de como é possível manipular imagens utilizando a biblioteca OpenCV, diferença entre imagens monocromáticas e policromáticas, a disposição de uma determinada imagem em forma de matrizes, sendo elas em tons de cinza ou na forma BGR, além de diversas operações básicas em imagens, importantes de se compreender antes de começar, de fato, o estudo de processamento de imagens.

3.1. IMAGENS MONOCROMÁTICAS

Uma imagem monocromática é uma função bidimensional contínua f(x,y), na qual x e y são coordenadas espaciais e o valor de f em qualquer ponto (x,y) é proporcional à intensidade luminosa (brilho ou nível de cinza) no ponto considerado. Como os computadores não são capazes de processar imagens contínuas, mas apenas arrays de números digitais, é necessário representar imagens como arranjos bidimensionais de pontos.

Cada ponto na grade bidimensional que representa a imagem digital é denominado elemento de imagem ou pixel. Na Fig. 2, apresenta-se a notação matricial usual para a localização de um pixel no arranjo de pixels de uma imagem bidimensional. O primeiro índice denota a posição da linha, m, na qual o pixel se encontra, enquanto o segundo, n, denota a posição da coluna. Se a imagem digital contiver M linhas e N colunas, o índice m variará de 0 a M-1, enquanto n variará de 0 a N-1. Observe-se o sentido de leitura (varredura) e a convenção usualmente adotada na representação espacial de uma imagem digital.

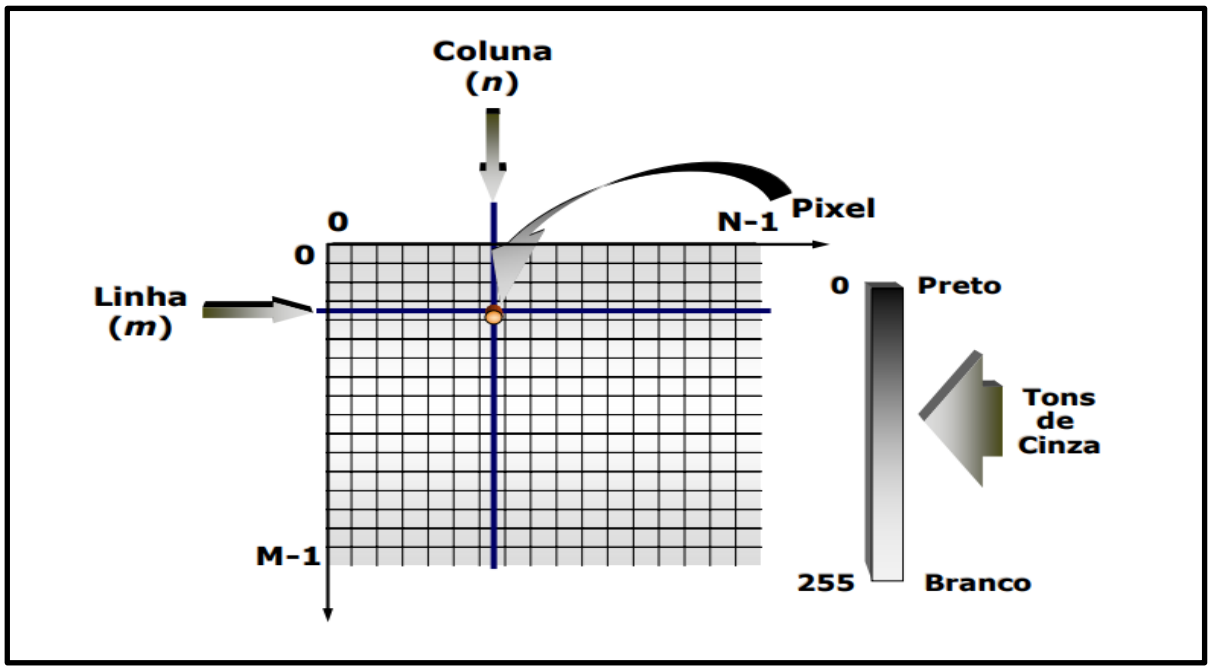

Fig. 2 – Representação de uma imagem digital bidimensional.

3.2. IMAGENS POLICROMÁTICAS

Em uma imagem digital colorida no sistema RGB, um pixel pode ser visto como um vetor cujas componentes representam as intensidades de vermelho, verde e azul de sua cor. A imagem colorida pode ser vista como a composição de três imagens monocromáticas, i.e.: f (x, y) = fR(x,y) + fG(x,y) + fB(x,y), na qual fR(x,y), fG(x,y), fB(x,y) representam, respectivamente, as intensidades luminosas das cores vermelha, verde e azul da imagem, no ponto (x,y). Assim, uma imagem colorida pode ser representada a partir de três matrizes, sendo uma para cada representação de cor.

Na Fig. 3, são apresentados os planos monocromáticos de uma imagem e o resultado da composição dos três planos. Os mesmos conceitos formulados para uma imagem digital monocromática aplicam-se a cada plano de uma imagem colorida.

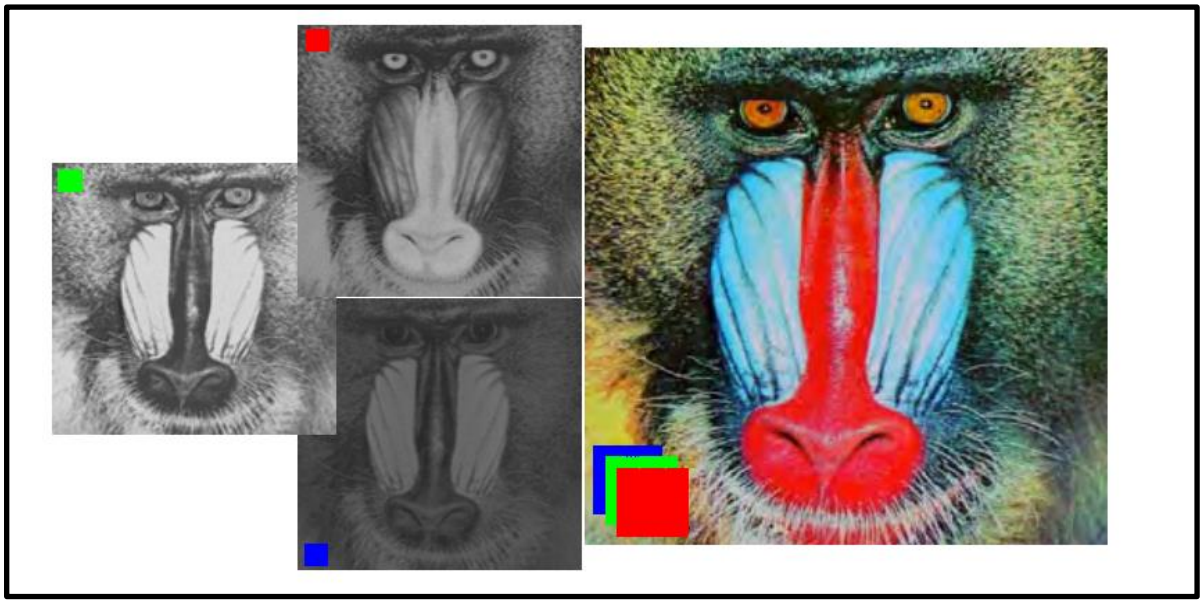

Fig. 3 – Representação de uma imagem digital bidimensional.

3.3. IMAGENS 2D E 3D

Uma imagem em duas dimensões (2D) pode ser definida como uma função bidimensional, f(x,y), em que x e y são coordenadas espaciais e a amplitude de f em qualquer coordenada (x,y) é chamada de intensidade ou nível de cinza da imagem, para esse ponto. Assim, quando x, y e f são quantidades finitas e discretas, chamamos de imagem digital (GONZALEZ; WOODS, 2006). Uma imagem também pode ser de três dimensões (3D), nesse caso, além de altura e largura, a imagem possui profundidade, o que leva sua manipulação a ser mais complexa que a de imagens 2D.

3.4. OPERAÇÕES COM IMAGENS

Este tópico fornece instruções sobre manipulações básicas de imagens, tais como ler e exibir imagens, transformar em escalas de cinza, etc, utilizando o OpenCV-Python.

3.4.1. OPENCV-PYTHON

O OpenCV-Python é a API do Python para OpenCV, projetada para resolver problemas de visão computacional, combinando as melhores qualidades da API OpenCV C++ e da linguagem Python.

Python é uma linguagem de programação de propósito geral iniciada por Guido van Rossum que se tornou muito popular muito rapidamente, principalmente por causa de sua simplicidade e legibilidade de código. Ele permite que o programador expresse ideias em menos linhas de código sem reduzir a legibilidade. Em comparação com linguagens como C/C++, o Python é mais lento. Dito isso, o Python pode ser facilmente estendido com o C/C++, o que nos permite escrever código computacionalmente intensivo em C/C++ e criar wrappers em Python que podem ser usados como módulos Python. Isso nos dá duas vantagens: primeiro, o código é tão rápido quanto o código C/C++ original (já que é o código C++ real trabalhando em segundo plano) e segundo, a codificação em Python é mais simples do que em C/C++. O OpenCV-Python é um wrapper do Python para a implementação original do OpenCV C++.

O OpenCV-Python utiliza o Numpy, que é uma biblioteca altamente otimizada para operações numéricas com uma sintaxe no estilo MATLAB. Todas as estruturas de matriz do OpenCV são convertidas para matrizes Numpy. Isso também facilita a integração com outras bibliotecas que usam o Numpy, como SciPy e Matplotlib.

3.4.2. COMEÇANDO COM IMAGENS

O tópicos 3.4.2 e seus subtópicos abordam as funções iniciais que englobam PDI, sendo demonstrados a teoria de cada operação e seu respectivos códigos implementados na linguagem python, sendo esclarecido os principais parâmetros das funções e métodos.

3.4.2.1. LER UMA IMAGEM

A função **cv.imread()** lê uma imagem. A imagem deve estar no diretório de trabalho ou um caminho completo da imagem deve ser fornecido. O primeiro argumento é o caminho da imagem, e o segundo argumento é referente a um sinalizador que especifica o modo como a imagem deve ser lida. Sinalizadores:

- **cv.IMREAD_COLOR**: Carrega uma imagem colorida. Qualquer transparência da imagem será negligenciada. É o sinalizador padrão.
- **cv.IMREAD_GRAYSCALE** : carrega a imagem no modo de escala de cinza.
- **cv.IMREAD_UNCHANGED** : carrega imagens como tal, incluindo canal alfa.

É possível referenciar os sinalizadores para valores inteiros 1, 0 ou -1, respectivamente.

3.4.2.2. EXIBIR UMA IMAGEM

A função **cv.imshow()** exibe uma imagem em uma janela. A janela se ajusta automaticamente ao tamanho da imagem. O primeiro argumento da função é o nome da janela, que é uma string. O segundo argumento é a imagem. Pode-se criar quantas janelas quiser, desde que tenham nomes diferentes.

cv.waitKey() é uma função de ligação ao teclado. Seu argumento é o tempo em milissegundos. A função aguarda por milissegundos especificados para qualquer evento de teclado. Quando pressionada qualquer tecla nesse tempo, o programa continua. Se **0** é passado, ele espera indefinidamente por um toque de tecla. Ele também pode ser configurado para detectar toques de tecla específicos.

3.4.2.3. SALVAR UMA IMAGEM

A função **cv.imwrite ()** salva uma imagem. O primeiro argumento do método é o nome do arquivo, o segundo é a imagem que você deseja salvar.

3.4.2.4. EXEMPLO DE CÓDIGO USANDO AS FUNÇÕES DE LER, EXIBIR E SALVAR IMAGENS

```
import numpy as np
import cv2 as cv
img = cv.imread('pavic.png')

cv.imshow('image',img)
k = cv.waitKey(0)
if k == 27:
   cv.destroyAllWindows()
elif k == ord('s'):
   cv.imwrite('pavic.png',img)
   cv.destroyAllWindows()
```

3.4.3. FUNÇÕES DE DESENHO NO OPENCV

É possível desenhar formas geométricas na imagem utilizando a biblioteca OpenCV.

3.4.3.1. LINHA

Para desenhar uma linha, é preciso passar as coordenadas iniciais e finais da linha. O seguinte algoritmo irá criar uma imagem preta e desenhar uma linha azul sobre ela, do canto superior esquerdo para o canto inferior direito.

```
import numpy as np
import cv2 as cv
img = np.zeros((512, 512, 3), np.uint8)
cv.line(img, (0, 0), (511, 511), (255, 0, 0), 5)
cv.imshow('image',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

3.4.3.2. RETÂNGULO

Para desenhar um retângulo, é preciso do canto superior esquerdo e do canto inferior direito do retângulo. Desta vez o algoritmo irá desenhar um retângulo verde no canto superior direito da imagem.

```
import numpy as np
import cv2 as cv
img = np.zeros((512, 512, 3), np.uint8)
cv.rectangle(img,(384,0),(510,128),(0,255,0),3)
cv.imshow('image',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

3.4.3.3. CÍRCULO

Para desenhar um círculo, você precisa das coordenadas do centro e do raio. Vamos desenhar um círculo dentro do retângulo desenhado acima.

```
import numpy as np
import cv2 as cv
img = np.zeros((512, 512, 3), np.uint8)
cv.circle(img,(447,63), 63, (0,0,255), -1)
cv.imshow('image',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

3.4.3.4. ELIPSE

Para desenhar a elipse, é necessário passar vários argumentos. Um argumento é a localização central (x, y). O próximo argumento é o comprimento dos

eixos (comprimento do eixo principal, comprimento do eixo menor). Ângulo é o ângulo de rotação da elipse no sentido anti-horário. startAngle e endAngle denotam o início e o fim do arco de elipse medido no sentido horário a partir do eixo principal.

```
import numpy as np
import cv2 as cv
img = np.zeros((512, 512, 3), np.uint8)
cv.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
cv.imshow('image',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

3.4.4. ATRIBUTOS DE IMAGENS

As propriedades da imagem incluem o número de linhas, colunas e canais, tipo de dados da imagem, número de pixels, etc.

3.4.4.1. FORMA DE UMA IMAGEM

A forma de uma imagem é acessada por **img.shape**. Ele retorna uma tupla de número de linhas, colunas e canais, Se uma imagem for em escala de cinza, a tupla retornada conterá apenas o número de linhas e colunas, portanto, é um bom método verificar se a imagem carregada é em tons de cinza ou em cores.

```
import numpy as np
import cv2 as cv
img = cv.imread('Pavic.PNG')
print(img.shape)
```

3.4.4.2. QUANTIDADE DE PIXELS DE UMA IMAGEM

É possível acessar a quantidade de pixels que uma determinada imagem possui utilizando o método **size**.

```
import cv2 as cv
img = cv.imread('Pavic.PNG')
print (img.size)
```

3.4.5. DIVISÃO E MESCLAGEM DE CANAIS

Às vezes pode ser necessário trabalhar separadamente os canais R, G e B de uma imagem. Nesse caso, é indicado dividir as imagens do BGR em canais únicos.

Em outros casos, talvez seja necessário associar esses canais individuais a uma imagem BGR.

```
import cv2 as cv
img = cv.imread('Pavic.PNG')
print (img.size)
b, g, r = cv.split (img)
img = cv.merge ((b, g, r))
cv.imshow('pavic',img)
cv.waitKey(0)
```

cv.split() é uma operação dispendiosa (em termos de tempo), portanto, em algumas situações, a utilização da indexação do Numpy é recomendada.

3.4.6. OPERAÇÕES ARITMÉTICAS EM IMAGENS

A função **cv.addWeighted** da OPenCV possibilita a junção de imagens, com diferentes pesos para cada um delas.

```
import cv2 as cv
img1 = cv.imread('PAVIC_1.png')
img2 = cv.imread('PAVIC_2.png')
dst = cv.addWeighted(img1,0.7, img2, 0, 3, 0)
cv.imshow('dst',dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

Exemplo de saída:

Fig. 4 – Representação de saída da função cv.addWeighted da OpenCV, utilizando pesos distintos para ambas as imagens.

Operações Bitwise (AND, OR, NOT e XOR) são muito utilizadas para extrair qualquer parte da imagem, definindo e trabalhando com ROI não retangular, etc.

Abaixo, é apresentado um exemplo de como alterar uma região específica de uma imagem utilizando as operações Bitwise.

```
# Carregar duas imagens
img1 = cv.imread('messi5.jpg')
img2 = cv.imread('opencv-logo-white.png')
# Colocar o logotipo no canto superior esquerdo, então é criado um
ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]
# Criar uma máscara de logo e criar uma máscara inversa também
img2gray = cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
ret, mask = cv.threshold(img2gray, 10, 255, cv.THRESH_BINARY)
mask_inv = cv.bitwise_not(mask)
# Apague a área do logotipo no ROI
img1_bg = cv.bitwise_and(roi,roi,mask = mask_inv)
# Leve apenas a região do logotipo da imagem do logotipo
img2_fg = cv.bitwise_and(img2,img2,mask = mask)
# Coloque o logotipo no ROI e modifique a imagem principal
dst = cv.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst
cv.imshow('res',img1)
cv.waitKey(0)
cv.destroyAllWindows()
```

Fig. 5 – Representação de saída de operações Bitwise da OpenCV, utilizando os operadores.

3.4.7. TÉCNICAS DE MEDIÇÃO E MELHORIA DE IMAGENS

No processamento de imagens, uma vez que você está lidando com um grande número de operações por segundo, é obrigatório que seu código não esteja apenas fornecendo a solução correta, mas também da maneira mais rápida. Neste tópico

você aprenderá sobre algumas funções capazes de medir e melhorar o desempenho do seu código.

3.4.7.1. MEDINDO O DESEMPENHO COM O OPENCV

A função **cv.getTickCount** retorna o número de ciclos de clock após um evento de referência (como o momento em que a máquina foi ligada) até o momento em que esta função é chamada. Então, se você chamá-lo antes e depois da execução da função, você obtém o número de ciclos de clock usados para executar uma função. A função **cv.getTickFrequency** retorna a freqüência de ciclos de clock, ou o número de ciclos de clock por segundo. Então, para encontrar o tempo de execução em segundos, basta subtrair o número inicial de ciclos de clock pelo final.

```
e1 = cv.getTickCount()
for i in range(5,49,2):
   # O range vai de 5 a 49, com passo 2
   img1 = cv.medianBlur(img1,i)
e2 = cv.getTickCount()
t = (e2 - e1)/cv.getTickFrequency()
print( t )
# Resultado: 0.521107655 segundos
```

3.4.7.2. TÉCNICAS DE OTIMIZAÇÃO DE DESEMPENHO

Existem várias técnicas e métodos de codificação para explorar o máximo desempenho do Python e do Numpy. A principal coisa a ser notada aqui é que, primeiro tente implementar o algoritmo de uma maneira simples. Quando estiver funcionando, faça o perfil dele, encontre os gargalos e otimize-os. Algumas dicas podem ser seguidas para alcançar o desempenho do código:

1. Evite usar loops no Python o máximo possível, especialmente loops duplos/triplos, etc. Eles são inerentemente lentos.
2. Vetorize o algoritmo/código na máxima extensão possível, pois o Numpy e o OpenCV são otimizados para operações vetoriais.
3. Explore a coerência do cache.
4. Nunca faça cópias do array, a menos que seja necessário, essa é uma operação dispendiosa.

Mesmo depois de fazer todas essas operações, se o seu código ainda estiver lento, ou o uso de loops grandes for inevitável, use bibliotecas adicionais como o Cython para torná-lo mais rápido.

3.5. TRANSFORMAÇÕES MORFOLÓGICAS

Morfologia digital ou matemática é uma modelagem destinada à descrição ou análise da forma de um objeto digital. O modelo morfológico para a análise de

imagens fundamenta-se na extração de informações a partir de transformações morfológicas, nos conceitos da álgebra booleana e na teoria dos conjuntos e reticulados. O princípio de morfologia digital se embasa no fato de que a imagem é um conjunto de pontos elementares (pixels ou voxels) que formam subconjuntos elementares bi ou tridimensionais. Os subconjuntos e a inter-relação entre eles formam estruturalmente a morfologia da imagem.

As operações básicas da morfologia digital são: a erosão, a partir da qual são removidos da imagem os pixels que não atendem a um dado padrão; e a dilatação, a partir da qual uma pequena área relacionada a um pixel é alterada para um dado padrão. Todavia, dependendo do tipo de imagem sendo processada (preto e branco, tons de cinza ou colorida) a definição destas operações muda, de forma que cada tipo deve ser considerado separadamente. As demais operações e transformações baseiam-se nos operadores básicos dos conjuntos, algumas interativas, e nos dois operadores básicos da morfologia matemática.

Seja a imagem da Fig. 6, na qual há dois objetos ou conjuntos de pixels A e B. Considere-se que os valores que os pixels podem assumir são binários, 0 ou 1, o que permite restringir a análise ao espaço discreto Z^2.

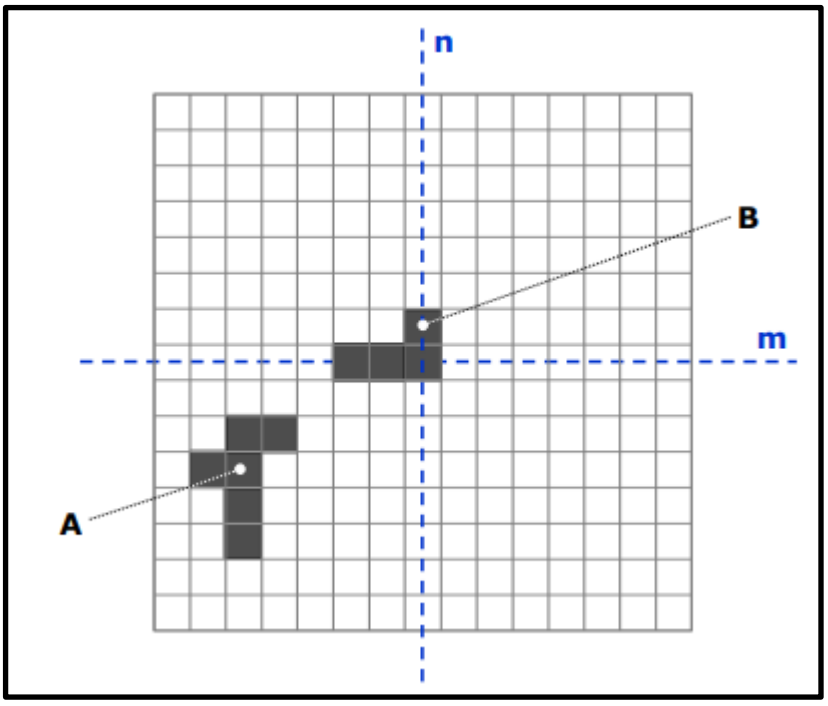

Fig. 6 - Imagem binária contendo 2 objetos, i.e.,2 conjuntos de pontos.

O objeto A consiste dos pontos α com pelo menos uma propriedade em comum, a saber:

Objeto A = {α ⊥ propriedade(α) = Verdade }

Assim sendo, o objeto B da Fig. 6 consistirá de {[-2,0][-1,0][0,0][0,1]}. O fundo

da imagem de A, denominado AC (complemento de A), consistirá de todos os pontos que não pertencem ao objeto A:

Fundo: A^c { $\alpha \perp \alpha \notin A$}

As operações fundamentais associadas com um objeto são o conjunto padrão de operações: união ({∪}), interseção ({∩}) e complemento ({C }) com translação. Dado um vetor x e um conjunto A, a translação, A + x, é definida como:

Translação: $A+ x = \{\alpha + x \perp \alpha \in A\Phi\}$

O conjunto básico de operações de Minkowski [12], adição e subtração, pode ser definido em função das considerações anteriores. Dados dois conjuntos A e B contidos em um conjunto C, a soma de Minkowski de A e B é o subconjunto de C, denotado $A\oplus$ B, dado por:

$A\oplus B = \{x\in C : \exists a\in A \text{ e } \exists b\in B, x = a + b\}$

A diferença de Minkowski entre A e B é o subconjunto de C, denotado A① B, dado por:

$A\Theta B = \{y \in C : \forall b\in B, (\exists a\in A, y = a - b\}$

Como mencionado anteriormente, as transformações singulares são realizadas através dos operadores elementares, de transformações de dilatação e erosão às quais foram incorporadas, posteriormente, mais duas transformações denominadas de anti-dilatação e anti-erosão. Dilatações e erosões são usadas para a criação de transformações mais sofisticadas, as quais conduzem a vários resultados relevantes quanto à análise de imagens, dentre os quais se citam os filtros morfológicos, o preenchimento de buracos, a extração de contornos e o reconhecimento de padrões. Os operadores de dilatação e erosão invariante por translação, sobre imagens binárias, advieram originalmente das operações de adição e subtração de Minkowski, cada um dos quais pode, em geral, ser caracterizado por um subconjunto denominado elemento estruturante.

Via de regra, a construção de sistemas morfológicos é implementada a partir da concepção do problema e da seleção dos operadores mais adequados à solução de interesse. A adequação de operadores constitui um dos grandes problemas encontrados na especificação dos elementos estruturantes. A criação de um mecanismo capaz de encontrar os elementos estruturantes “adequados” à realização da transformação de interesse é uma possível solução para tal problema. A partir das operações básicas de Minkowski, podem-se definir as operações básicas da morfologia matemática, dilatação e erosão:

Dilatação: $D(A, B) = A \oplus B = A \oplus B = \{x \in E \mid Bx \cap A \neq \varnothing\}$

Erosão: $E(A, B) = A \ominus B = \{x \in E \mid Bx \subseteq A\}$

Tanto o conjunto A quanto o conjunto B podem ser considerados como sendo imagens. Todavia, A costuma ser considerado como sendo a imagem sob análise e B como o elemento estruturante, o qual está para a morfologia como a máscara (mask, template ou kernel) está para teoria de filtragem linear. Os elementos estruturantes mais comuns são os conjuntos 4-conexões e 8-conexões, N4 e N8 (Fig. 7).

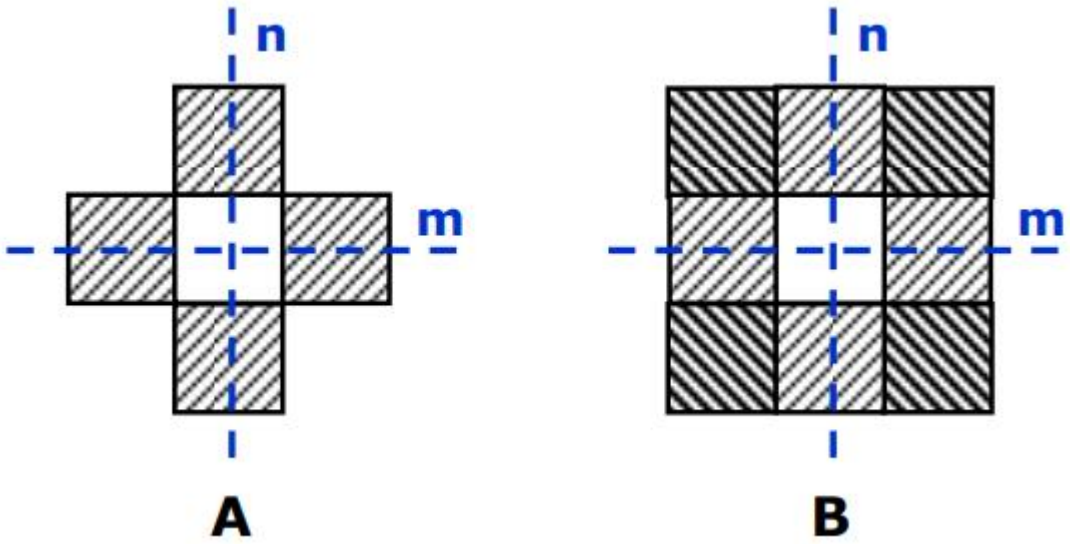

Fig. 7 - Elementos estruturantes: (A) padrão N4; e (B) padrão N8.

A dilatação, em geral, faz com que o objeto cresça no tamanho. Buracos menores do que o elemento estruturante são eliminados e o número de componentes pode diminuir. Por sua vez, a erosão reduz as dimensões do objeto. Objetos menores do que o elemento estruturante são eliminados e o número de componentes pode aumentar. O modo e a magnitude da expansão ou redução da imagem dependem necessariamente do elemento estruturante B. A aplicação de uma transformação de dilatação ou erosão a uma imagem sem a especificação de um elemento estruturante, não produzirá nenhum efeito.

3.5.1. EROSÃO

A idéia básica da erosão é apenas como a erosão do solo, ela elimina os limites do objeto em primeiro plano (sempre tente manter o primeiro plano em branco). O kernel desliza pela imagem (como na convolução 2D). Um pixel na imagem original (1 ou 0) será considerado 1 somente se todos os pixels sob o kernel forem 1, caso contrário, ele será erodido (reduzido a zero).

Então, o que acontece é que todos os pixels próximos ao limite serão descartados, dependendo do tamanho do kernel. Assim, a espessura ou o tamanho

do objeto de primeiro plano diminui ou simplesmente a região branca diminui na imagem. É útil para remover pequenos ruídos brancos, destacar dois objetos conectados, etc.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
cv.imshow("erosion",erosion)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 8 - entrada e saída de uma imagem no método de erosão.

3.5.2. DILATAÇÃO

Dilatação é o oposto da erosão. Aqui, um elemento de pixel é '1' se pelo menos um pixel sob o kernel for '1'. Por isso, aumenta a região branca na imagem ou o tamanho do objeto em primeiro plano aumenta. Normalmente, em casos como a remoção de ruídos, a erosão é seguida por dilatação. Porque a erosão remove ruídos brancos, mas também encolhe nosso objeto. Então nós dilatamos isso. Como o ruído se foi, eles não voltarão, mas a área de objeto aumenta. Também é útil para unir partes quebradas de um objeto.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
dilation = cv.dilate(img,kernel,iterations = 1)
cv.imshow("dilatação",dilation)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 9 - entrada e saída de uma imagem no método de dilatação.

3.5.3. ABERTURA

A abertura é apenas outro nome de erosão seguido de dilatação. É útil na remoção de ruído, como explicamos acima. Aqui nós usamos a função cv.morphologyEx().

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
abertura = cv.morphologyEx (img, cv.MORPH_OPEN, kernel)
cv.imshow("abertura",abertura)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 10 - entrada e saída de uma imagem no método de abertura.

3.5.4. ENCERRAMENTO

O fechamento é o inverso da abertura, dilatação seguida de erosão. É útil para fechar pequenos furos dentro dos objetos em primeiro plano, ou pequenos pontos pretos no objeto.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
```

```
closing = cv.morphologyEx (img, cv.MORPH_CLOSE, kernel)
cv.imshow("closing",closing)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 11 - entrada e saída de uma imagem no método de encerramento.

3.5.5. GRADIENTE MORFOLÓGICO

É a diferença entre dilatação e erosão de uma imagem. O resultado será parecido com o contorno do objeto.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
gradient = cv.morphologyEx (img, cv.MORPH_GRADIENT, kernel)
cv.imshow("gradient",gradient)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 12 - entrada e saída de uma imagem no método de gradiente morfológico.

3.5.6. CARTOLA

É a diferença entre a imagem de entrada e a abertura da imagem. Abaixo o exemplo é feito para um kernel 9x9.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
tophat = cv.morphologyEx (img, cv.MORPH_TOPHAT, kernel)
cv.imshow("tophat",tophat)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 13 - entrada e saída de uma imagem no método cartola.

3.5.7. BLACK HAT

É a diferença entre o fechamento da imagem de entrada e a imagem de entrada.

```
import cv2 as cv
import numpy as np
img = cv.imread('PAVIC.png',0)
kernel = np.ones((5,5),np.uint8)
blackhat = cv.morphologyEx (img, cv.MORPH_BLACKHAT, kernel)
cv.imshow("blackhat",blackhat)
cv.imshow("img",img)
cv.waitKey(0)
```

Fig 14 - entrada e saída de uma imagem no método black hat.

4. HISTOGRAMA

O histograma pode ser considerado um gráfico, o que lhe dá uma idéia geral sobre a distribuição de intensidade de uma imagem. É um gráfico com valores de pixel (variando de 0 a 255, nem sempre).

É apenas outra maneira de entender a imagem. Ao olhar para o histograma de uma imagem, obtém-se a intuição sobre contraste, brilho, distribuição de intensidade, dentre outras características dessa imagem. Inúmeras ferramentas de processamento de imagens fornecem recursos no histograma. Abaixo está uma imagem para melhor entendimento.

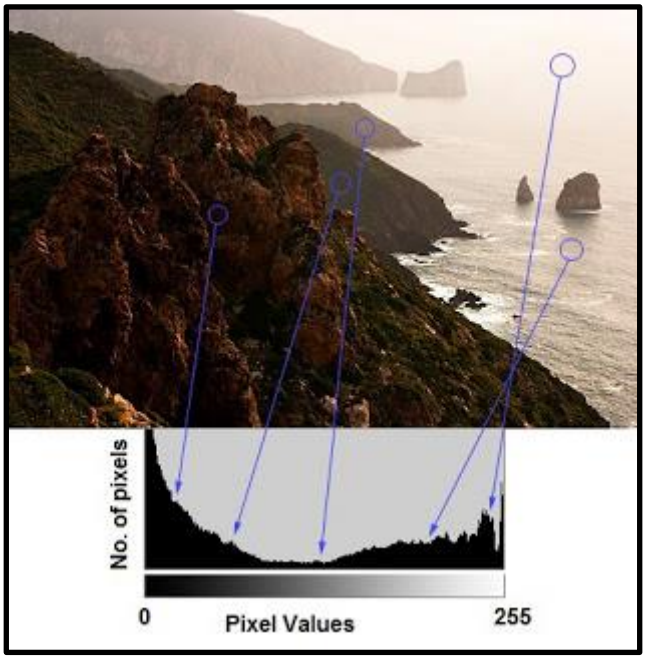

Fig. 15 – Representação de um histograma com seus valores relacionados a uma imagem.

Este histograma é desenhado para imagem em escala de cinza, não para imagem colorida. A região esquerda do histograma mostra a quantidade de pixels mais escuros na imagem e a região direita mostra a quantidade de pixels mais claros. A partir do histograma, pode-se observar que a região escura é mais do que uma região mais brilhante e a quantidade de tons médios (valores de pixels na faixa média, digamos, cerca de 127) é muito menor.

4.1. ANÁLISE DE HISTOGRAMA

O histograma acima mostra o número de pixels para cada valor de pixel, ou seja, de 0 a 255, ou seja, é necessário 256 valores para mostrar o histograma acima. Mas, considere não precisar encontrar o número de pixels para todos os valores de pixel separadamente, mas o número de pixels em um intervalo de valores de pixel? Por exemplo, é necessário encontrar o número de pixels entre 0 e 15, depois 16 a 31,

..., 240 a 255. Para isso precisará apenas de 16 valores para representar o histograma.

Então, simplesmente é dividido o histograma inteiro em 16 sub-partes e o valor de cada sub-parte é a soma de todas as contagens de pixels. Cada sub-parte é chamada "BIN". No primeiro caso, o número de bins era 256 (um para cada pixel) enquanto no segundo caso, era apenas 16. BINS é representado pelo termo histSize em documentos do OpenCV.

DIMS: É o número de parâmetros para os quais coletamos os dados. Neste caso, coletamos dados referentes a apenas uma coisa, valor de intensidade.

RANGE: É o intervalo de valores de intensidade que você deseja medir. Normalmente, é [0,256], ou seja, todos os valores de intensidade.

4.1.1. CÁLCULO DO HISTOGRAMA NO OPENCV

A função **cv.calcHist()** é utilizada para produzir o histograma. Seus parâmetros são:

- imagem: é a imagem de origem do tipo uint8 ou float32, deve ser dado entre colchetes, ou seja, "[img]".
- canal: também é dado em colchetes. É o índice do canal para o qual calculamos o histograma. Por exemplo, se a entrada for imagem em tons de cinza, seu valor será [0]. Para a imagem colorida, você pode passar [0], [1] ou [2] para calcular o histograma do canal azul, verde ou vermelho, respectivamente.
- máscara: imagem da máscara. Para encontrar o histograma da imagem completa, é dado como "None". Mas se a intenção é encontrar o histograma de uma determinada região da imagem, é necessário criar uma imagem de máscara para isso e dar como máscara.
- histSize: representa a nossa contagem de BIN. Precisa ser dado entre colchetes. Para a escala completa, passamos [256].
- intervalos: esta é a nossa gama. Normalmente, é [0,256].

```
img = cv.imread('PAVIC.png',0)
hist = cv.calcHist([img], [0], None, [256], [0,256])
```

4.1.2. CÁLCULO DO HISTOGRAMA EM NUMPY

Numpy também fornece uma função, **np.histogram()**. Então, ao invés da função **calcHist()** , é possível utilizar o **np.histogram().**

Hist é o mesmo que calculamos antes. Mas as caixas terão 257 elementos, porque a Numpy calcula as bandejas como 0-0,99, 1-1,99, 2-2,99, etc. Portanto, o intervalo final seria de 255-255,99. Para representar isso, eles também adicionam 256 no final das caixas. Mas não precisamos disso 256. Até 255 é suficiente.

```
hist, bins= np.histograma (img.ravel (), 256, [0,256])
```

4.1.3. USANDO O MATPLOTLIB

O Matplotlib vem com uma função de plotagem de histograma: **matplotlib.pyplot.hist()**.

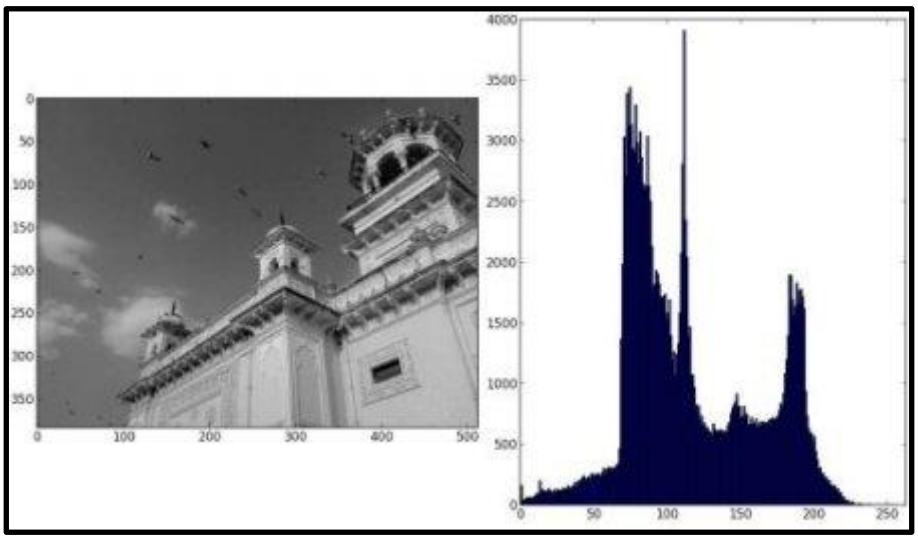

Fig. 16 – Representação de um histograma com seus valores relacionados a uma imagem.

4.2. EQUALIZAÇÃO DO HISTOGRAMA

Considere uma imagem cujos valores de pixel estão limitados apenas a um intervalo específico de valores. Por exemplo, uma imagem mais clara terá todos os pixels confinados a valores altos. Mas uma boa imagem terá pixels de todas as regiões da imagem. Então é necessário esticar este histograma para qualquer um dos fins, e é isso que o Equalização do Histograma faz (em palavras simples). Isso normalmente melhora o contraste da imagem.

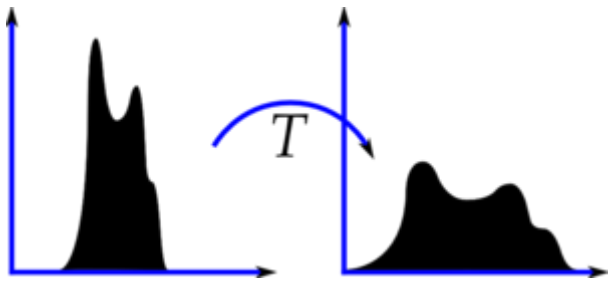

Fig. 17 - Exemplo de equalização do histograma

O OpenCV tem uma função para fazer equalização, **cv.equalizeHist()**. Sua entrada é apenas uma imagem em tons de cinza e a saída é uma imagem equalizada do histograma. Abaixo está um pequeno snippet de código mostrando seu uso.

```
img                    =                    cv.imread('pavic.jpg',0)
equ                      =                      cv.equalizeHist(img)
res   =   np.hstack((img,equ))   #   empilhando   imagens   lado   a
ladocv.imwrite('res.png',res)
```

Fig. 18 - Exemplo de entrada e saída em equalização do histograma

4.3. HISTOGRAMA 2D

No tópico anterior, calculamos e plotamos o histograma unidimensional. É chamado unidimensional porque estamos considerando apenas um recurso, ou seja, o valor da intensidade da escala de cinza do pixel. Mas nos histogramas bidimensionais, é necessário considerar dois recursos. Normalmente, ele é usado para produzir histogramas de cores, onde dois recursos são valores de matiz e saturação de cada pixel.

Para histogramas de cores, precisamos converter a imagem de BGR para HSV. (Lembre-se, para o histograma 1D, convertemos de BGR para Grayscale). Para histogramas 2D, seus parâmetros serão modificados da seguinte forma:

- channels = [0,1] porque precisamos processar os planos H e S.
- bins = [180,256] 180 para o plano H e 256 para o plano S.
- range = [0,180,0,256] O valor de Hue está entre 0 e 180 e a Saturação está entre 0 e 256.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg')
hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
hist = cv.calcHist( [hsv], [0, 1], None, [180, 256], [0, 180, 0,
256] )
```

```
plt.imshow(hist,interpolation = 'nearest')
plt.show()
```

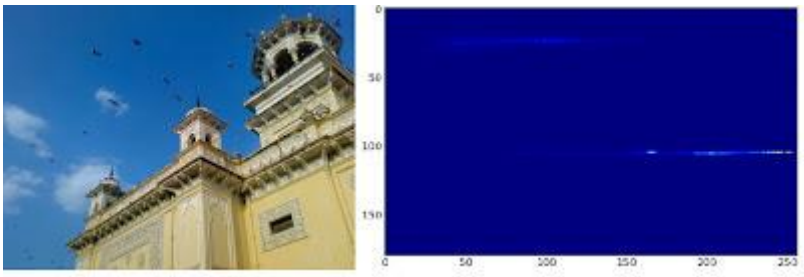

Fig. 19 - Exemplo de histograma 2D

4.4. RETROPROJEÇÃO DO HISTOGRAMA

Retroprojeção cria uma imagem do mesmo tamanho (mas canal único) como a da imagem de entrada, onde cada pixel corresponde à probabilidade daquele pixel pertencer ao objeto estudado. Em mundos mais simples, a imagem de saída terá o objeto de interesse mais branco em comparação com a parte restante. Bem, isso é uma explicação intuitiva. A retroprojeção de histograma é usada com o algoritmo de camshift.

Para realizar a retroprojeção deve-se fazer com que cada histograma de uma imagem contenha o objeto de interesse. O objeto deve preencher a imagem o máximo possível para obter melhores resultados. E um histograma de cores é preferível ao histograma em escala de cinza, porque a cor do objeto é uma maneira melhor de definir o objeto do que a intensidade da escala de cinza. Então "projetamos de volta" este histograma sobre a imagem de teste, onde é preciso encontrar o objeto, ou seja, calculamos a probabilidade de cada pixel pertencente ao solo e o mostramos. A saída resultante no limiar adequado nos dá o solo sozinho.

O OpenCV fornece uma função embutida **cv.calcBackProject()**. Seus parâmetros são quase os mesmos que a função **cv.calcHist()**, um deles é o histograma, que é o histograma do objeto e temos que encontrá-lo. Além disso, o histograma do objeto deve ser normalizado antes de passar para a função backproject, que retorna a imagem de probabilidade. Então envolvemos a imagem com um kernel disc e aplicamos o threshold. Abaixo estão o código e a saída:

```
import numpy as np
import cv2 as cv
roi = cv.imread(pavic.png')
hsv = cv.cvtColor(roi,cv.COLOR_BGR2HSV)
target = cv.imread(pavic2.png')
```

```
hsvt = cv.cvtColor(target,cv.COLOR_BGR2HSV)
# histograma de cálculo de objetos
roihist = cv.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0,
256] )
# normalize o histograma e aplique a retroprojeção
cv.normalize(roihist,roihist,0,255,cv.NORM_MINMAX)
dst = cv.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)
# Agora convolute com disco circulardisc =
cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
cv.filter2D(dst,-1,disc,dst)
# limiar e binário and
ret,thresh = cv.threshold(dst,50,255,0)
thresh = cv.merge((thresh,thresh,thresh))
res = cv.bitwise_and(target,thresh)
res = np.vstack((target,thresh,res))
cv.imwrite('res.jpg',res)
```

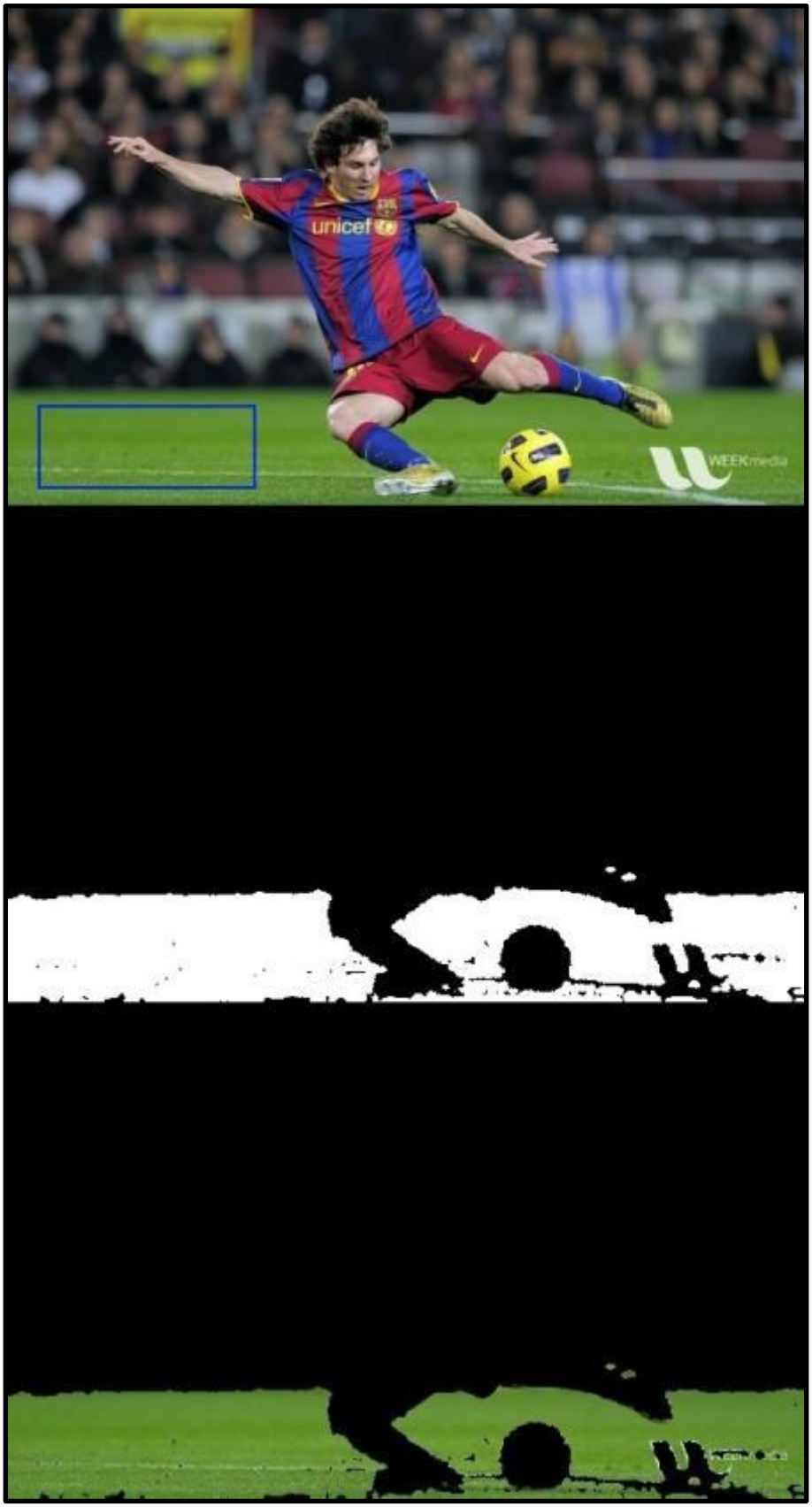

Fig. 20 - Exemplo de entrada e saída em retroprojeção do histograma.

5. PRÉ-PROCESSAMENTO

Pré-processamento refere-se ao processamento inicial de dados brutos para calibração radiométrica da imagem, correção de distorções geométricas e remoção de ruído. Os subtópicos abaixo definem e exemplificam algumas das técnicas utilizadas nesta fase do processamento de imagens, como transformações geométricas (redimensionamento, mudança, rotação, etc.) e suavização de imagens.

5.1. TRANSFORMAÇÕES GEOMÉTRICAS DE IMAGENS

Neste tópico você aprenderá como aplicar diferentes transformações geométricas a imagens, como tradução, rotação, transformação afim, etc.

5.1.1. ESCALA

O OpenCV vem com a função **cv.resize()**, que faz o redimensionamento de uma imagem. O tamanho da imagem pode ser especificado manualmente ou você pode especificar o fator de escala. Métodos de interpolação diferentes são usados. Os métodos de interpolação preferidos são **cv.INTER_AREA** para redução e **cv.INTER_CUBIC** e **cv.INTER_LINEAR** para zoom. Por padrão, o método de interpolação usado é **cv.INTER_LINEAR** para todos os propósitos de redimensionamento. O código abaixo exemplifica o uso de um método de interpolação para redimensionar uma imagem:

```
import cv2 as cv
img = cv.imread('pavic.jpg')
height = img.shape[0]
width = img.shape[1]

res = cv.resize(img,(2*width, 2*height), interpolation =
cv.INTER_CUBIC)
cv.imshow('res', res)
cv.waitKey(0)
```

5.1.2. TRADUÇÃO

Neste tópico, vamos discutir sobre a mudança de uma imagem. Digamos que queremos mover a imagem dentro do nosso quadro de referência. Na terminologia de visão computacional, isso é chamado de tradução. Podemos fazer isso através do seguinte algoritmo:

```
import matplotlib.pyplot as plt1
```

```
import numpy as np
import cv2 as cv
img = cv.imread('lampada.png',1)
rows = img.shape[0]
cols = img.shape[1]
M = np.float32([[1,0,100],[0,1,50]])
dst = cv.warpAffine(img,M,(cols,rows))
plt1.subplot (121), plt1.imshow (img), plt1.title ( 'Input' )
plt1.subplot (122), plt1.imshow (dst), plt1.title ( 'Saída' )
plt1.show()
```

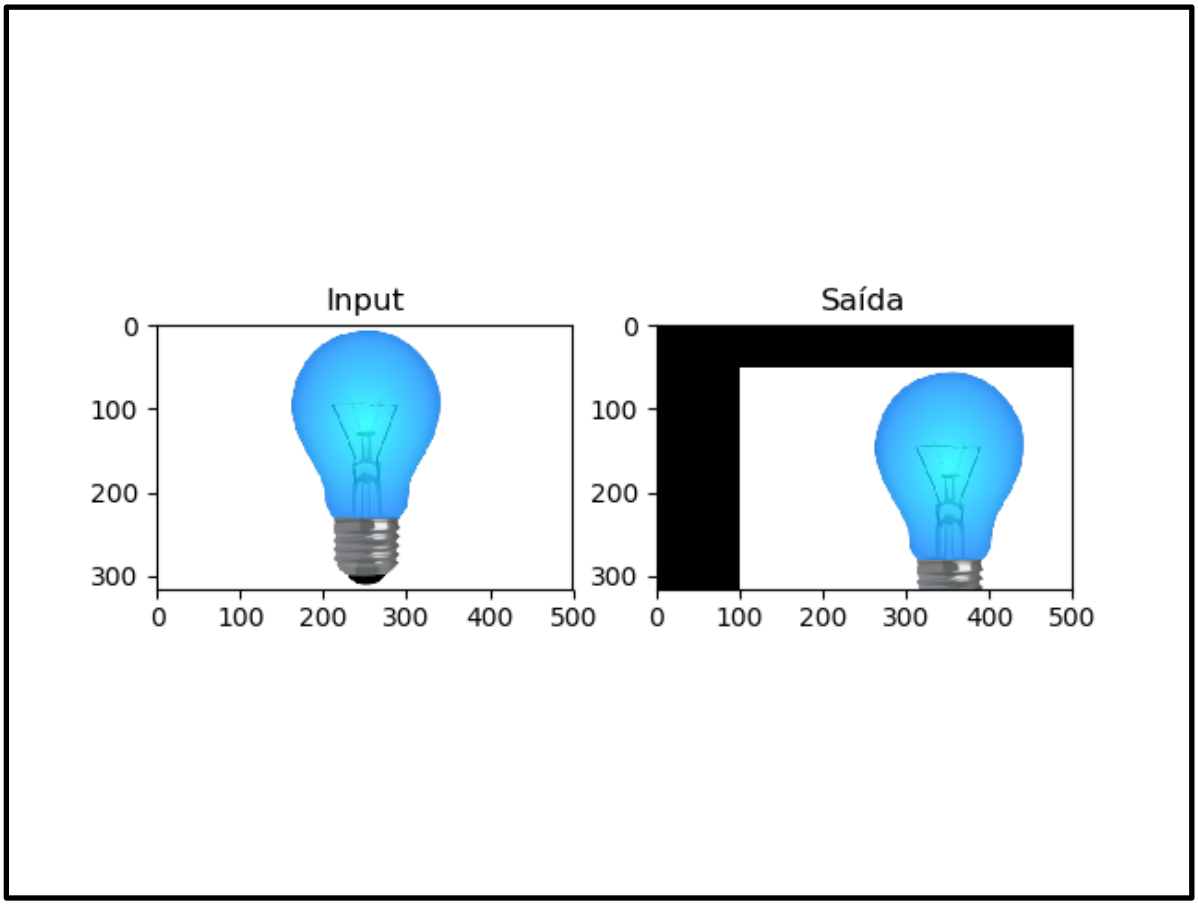

Fig. 21 - entrada e saída do código de tradução de imagens

Atenção: o terceiro argumento da função **cv2.warpAffine()** é o tamanho da imagem de saída, que deve estar na forma de**(largura, altura)**. Lembre-se: largura = número de colunas e altura = número de linhas.

5.1.3. ROTAÇÃO

A rotação também é uma forma de transformação e podemos alcançá-la usando a seguinte matriz de transformação:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Para encontrar essa matriz de transformação, o OpenCV fornece uma função, **cv.getRotationMatrix2D**. Verifique abaixo o exemplo que gira a imagem em 90 graus em relação ao centro sem qualquer escala:

```
import cv2 as cv
import matplotlib.pyplot as plt1
img = cv.imread('logo.jpeg',0)
rows = img.shape[0]
cols = img.shape[1]
# cols-1 e rows-1 são os limites das coordenadas.
M = cv.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1)
dst = cv.warpAffine(img,M,(cols,rows))
plt1.subplot(121),plt1.imshow(img),plt1.title('Input')
plt1.subplot(122),plt1.imshow(dst),plt1.title('Output')
plt1.show()
cv.waitKey(0)
```

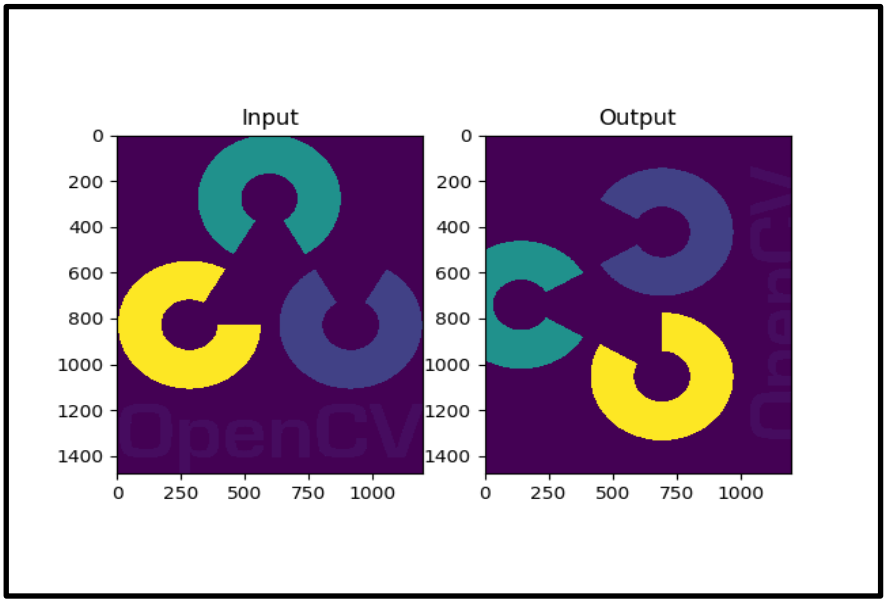

Fig. 22 - entrada e saída do código de rotação de imagens

5.1.4. TRANSFORMAÇÃO AFIM

Em muitos sistemas de imagem, as imagens detectadas estão sujeitas a distorção geométrica introduzida por irregularidades de perspectiva, em que a posição da (s) câmera (s) em relação à cena altera as dimensões aparentes da geometria da cena. A aplicação de uma transformação afim a uma imagem uniformemente distorcida pode corrigir uma série de distorções de perspectiva, transformando as medidas das coordenadas ideais para aquelas realmente usadas. (Por exemplo, isso é útil em imagens de satélite em que mapas de terra geometricamente corretos são desejados).

A transformação afim em uma imagem pode ser feita através do seguinte código:

```
import cv2 as cv
import numpy as np
import matplotlib as plt
import matplotlib.pyplot as plt1
img = cv.imread('logo.jpeg')
rows = img.shape[0]
cols = img.shape[1]
ch = img.shape[2]
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2)
dst = cv.warpAffine(img,M,(cols,rows))
plt1.subplot(121),plt1.imshow(img),plt1.title('Input')
plt1.subplot(122),plt1.imshow(dst),plt1.title('Output')
plt1.how(s)
```

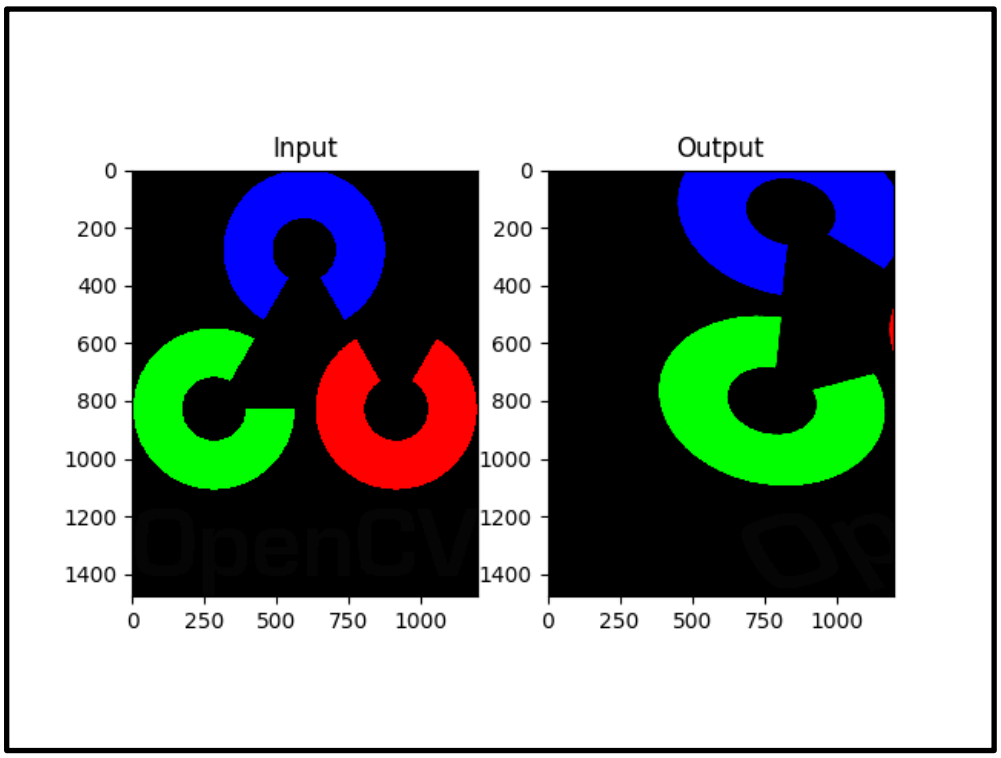

Fig. 23 - entrada e saída do código de transformação afim

5.1.5. TRANSFORMAÇÃO PERSPECTIVA

No geral, a transformação da perspectiva lida com a conversão do mundo 3D em imagem 2D. O mesmo princípio no qual a visão humana funciona e o mesmo princípio no qual a câmera funciona. Para transformação de perspectiva, você precisa de uma matriz de transformação 3x3. Linhas retas permanecerão retas mesmo após a transformação. Para encontrar essa matriz de transformação, você precisa de 4 pontos na imagem de entrada e pontos correspondentes na imagem de saída. Entre esses 4 pontos, 3 deles não devem ser colineares. Em seguida, a matriz de transformação pode ser produzida pela função **cv.getPerspectiveTransform**. Em seguida, aplique **cv.warpPerspective** com essa matriz de transformação 3x3.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt1
img = cv.imread ( 'logo.jpeg' )
rows = img.shape[0]
cols = img.shape[1]
ch = img.shape[2]
pts1 = np.float32([[50,50],[200,50],[50,200]])
```

```
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2)
dst = cv.warpAffine(img,M,(cols,rows))
plt1.subplot(121),plt1.imshow(img),plt1.title('Input')
plt1.subplot(122),plt1.imshow(dst),plt1.title('Output')
plt1.show()
```

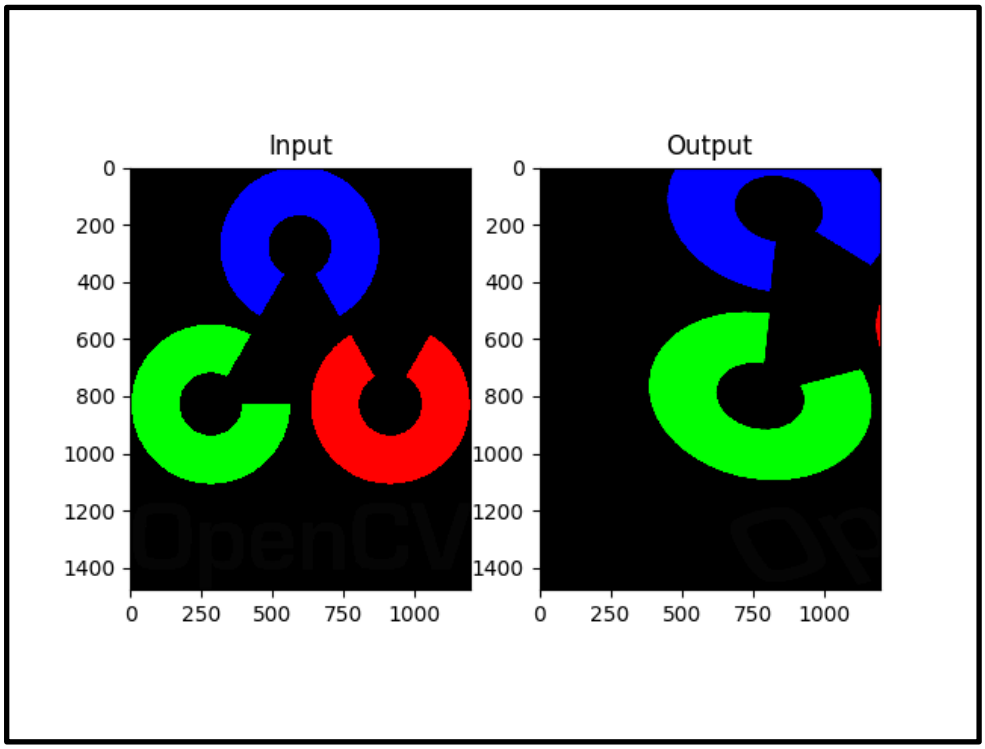

Fig. 24 - entrada e saída do código de transformação perspectiva

5.2. SUAVIZAÇÃO DE IMAGENS

O desfoque da imagem é obtido pela convolução da imagem com um kernel de filtro de baixa passagem. É útil para remover o ruído. Ele realmente remove o conteúdo de alta freqüência (por exemplo: ruído, bordas) da imagem, resultando em bordas sendo borradas quando este filtro é aplicado. (Bem, existem técnicas de desfoque que não desfocam as bordas). O OpenCV fornece principalmente quatro tipos de técnicas de desfoque.

5.2.1. MÉDIA

Isso é feito através da convolução da imagem com um filtro de caixa normalizado. Ele simplesmente pega a média de todos os pixels sob a área do kernel e substitui o elemento central por essa média. Isso é feito pela função **cv2.blur()** ou

cv2.boxFilter(). Devemos especificar a largura e a altura do kernel. Um filtro de caixa normalizado 3x3 ficaria assim:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Verifique o exemplo de demonstração abaixo com um kernel de tamanho 5x5:

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('ufpi.jpg')
blur = cv2.blur(img,(5,5))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Borrado')
plt.xticks([]), plt.yticks([])
plt.show()
```

Saída:

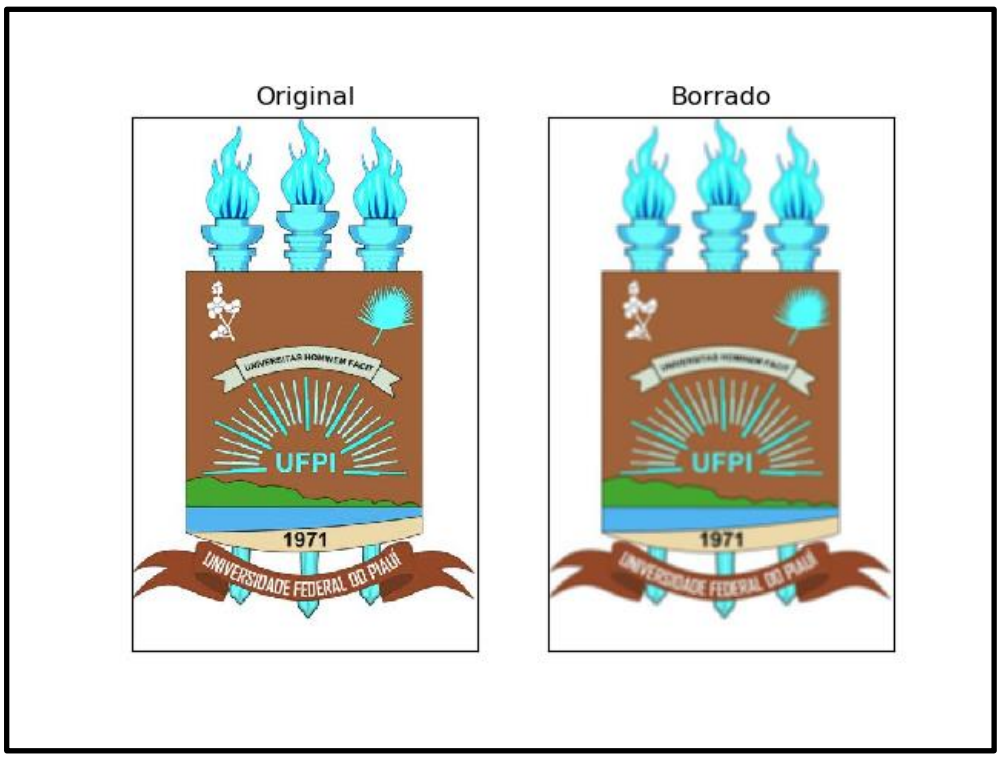

Fig. 25 - entrada e saída do código de média

5.2.2. FILTRAGEM GAUSSIANA

Nesta abordagem, em vez de um filtro de caixa consistindo em coeficientes de filtro iguais, um kernel gaussiano é usado. Isso é feito com a função cv2.GaussianBlur() . Devemos especificar a largura e altura do kernel que deve ser positivo e ímpar. Também devemos especificar o desvio padrão nas direções X e Y, sigmaX e sigmaY, respectivamente. Se somente sigmaX for especificado, sigmaY é considerado igual a sigmaX. Se ambos forem dados como zeros, eles serão calculados a partir do tamanho do kernel. A filtragem gaussiana é altamente eficaz na remoção do ruído gaussiano da imagem.

O código acima pode ser modificado para desfocagem gaussiana, ficando da seguinte forma:

```
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('ufpi.jpg')
blur = cv2.GaussianBlur(img,(5,5),0)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Borrado')
plt.xticks([]), plt.yticks([])
plt.show()
```

Saída:

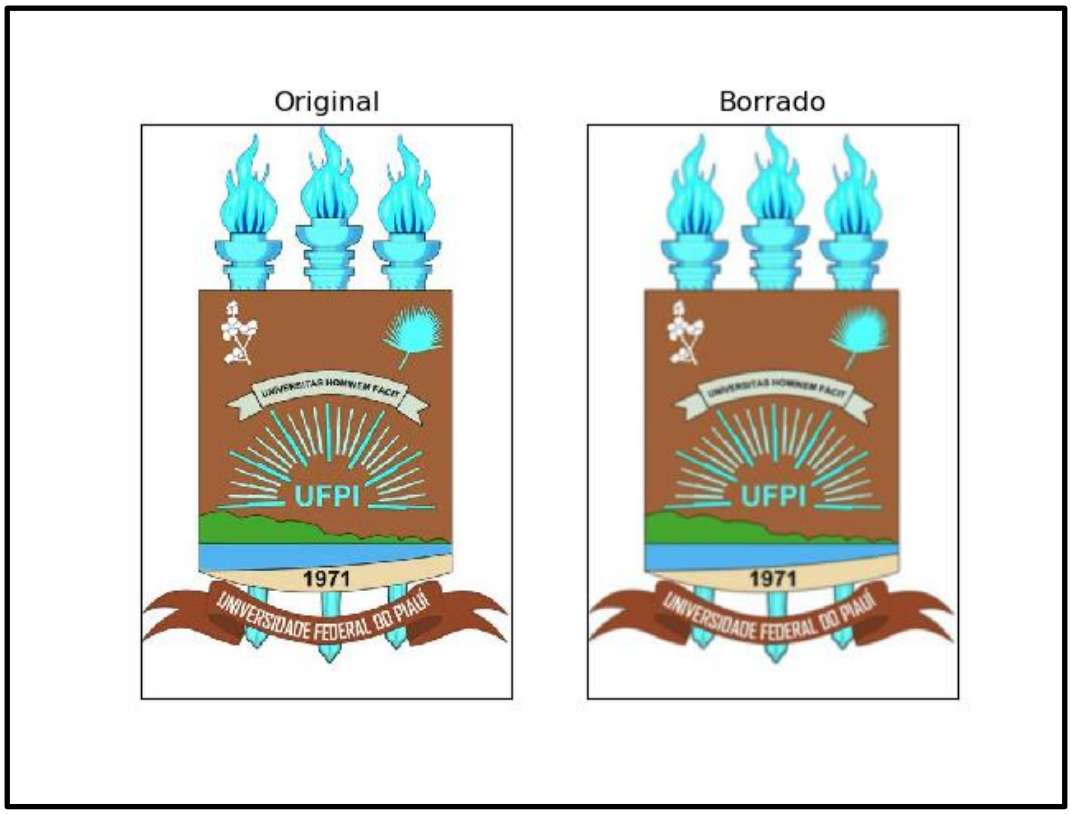

Fig. 26 - entrada e saída do código de filtragem Gaussiana

5.2.3. FILTRAGEM MEDIANA

Aqui, a função cv2.medianBlur() calcula a mediana de todos os pixels sob a janela do kernel e o pixel central é substituído por este valor mediano. Uma coisa interessante a notar é que, nos filtros gaussiano e de caixa, o valor filtrado para o elemento central pode ser um valor que pode não existir na imagem original. No entanto, este não é o caso na filtragem mediana, uma vez que o elemento central é sempre substituído por algum valor de pixel na imagem. Isso reduz o ruído de forma eficaz. O tamanho do kernel deve ser um inteiro ímpar positivo.

Nesta demonstração, adicionamos um ruído de 50% à nossa imagem original e usamos um filtro mediano. Verifique o código e o resultado:

```
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('opencv_logo.png')
mediana = cv2.medianBlur(img,5)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(mediana),plt.title('Mediana')
plt.xticks([]), plt.yticks([])
plt.show()
```

Saída:

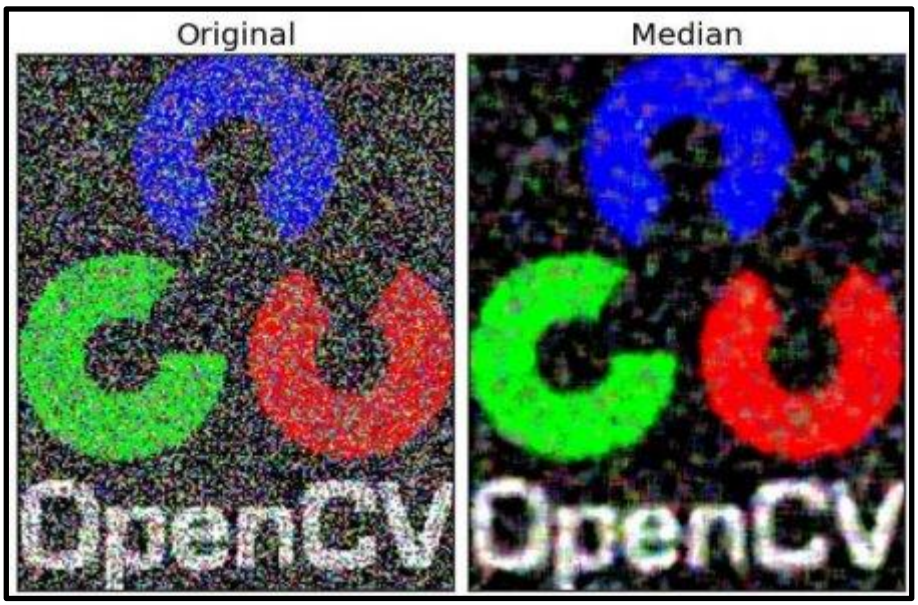

Fig. 27 - entrada e saída do código de filtragem mediana

5.2.4. FILTRAGEM BILATERAL

Como observamos, os filtros que apresentamos anteriormente tendem a borrar as bordas. Este não é o caso para o filtro bilateral, cv2.bilateralFilter(), que foi definido para a remoção de ruído, preservando as bordas. Mas a operação é mais lenta em comparação com outros filtros. Já vimos que um filtro gaussiano leva a vizinhança ao redor do pixel e encontra sua média ponderada gaussiana. Este filtro gaussiano é uma função apenas do espaço, isto é, os pixels próximos são considerados durante a filtragem. Não considera se os pixels têm quase o mesmo valor de intensidade e não considera se o pixel está em uma borda ou não. O efeito resultante é que os filtros gaussianos tendem a borrar as bordas, o que é indesejável.

O filtro bilateral também usa um filtro gaussiano no domínio espacial, mas também usa mais um componente (multiplicativo) de filtro gaussiano que é uma função das diferenças de intensidade de pixel. Observe o código e o exemplo de saída da filtragem bilateral:

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('ruido.jpg')

blur = cv2.bilateralFilter(img,9,75,75)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Borrado')
plt.xticks([]), plt.yticks([])
plt.show()
```

Saída:

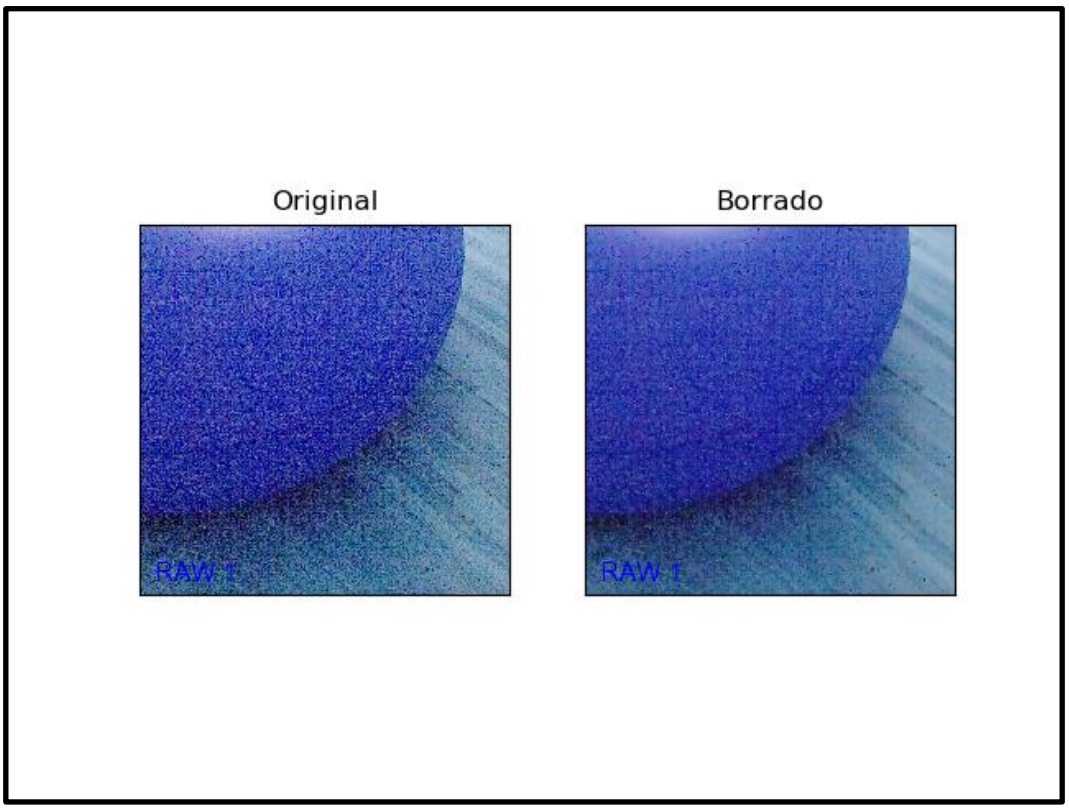

Fig. 28 - entrada e saída do código de filtragem bilateral

6. SEGMENTAÇÃO

A segmentação consiste na primeira etapa de processamento da imagem quando consideramos uma análise do ponto de vista da INFORMAÇÃO nela presente. O termo segmentação vem do termo em inglês "image segmentation", criado durante os anos 80.

Esta área representa até hoje uma linha de pesquisa importante do processamento de imagens, principalmente por ela estar na base de todo o processamento da informação em uma imagem. Segmentar consiste na realidade em dividir a imagem em diferentes regiões, que serão posteriormente analisadas por algoritmos especializados em busca de informações ditas de "alto-nível".

A imagem obtida é composta por apenas duas regiões, por exemplo uma região branca (fundo) e outra preta (células/objeto). Esta imagem, com 2 níveis de cinza, é conhecida como Imagem Binária.

Devido às grandes facilidades na manipulação deste tipo de imagens, principalmente porque reduzimos significativamente a quantidade de dados, elas são frequentemente utilizadas no processo de tratamento da informação. Existem diversas técnicas de segmentação de imagens, mas não existe nenhum método único que seja capaz de segmentar todos os tipos de imagem.

Globalmente, uma imagem em níveis de cinza pode ser segmentada de duas maneiras: ou consideramos a semelhança entre os níveis de cinza ou consideramos as sua diferenças.

A detecção de um contorno de um objeto, através de matrizes do tipo Passa-Alta, é um exemplo de técnicas baseado nas diferenças. Neste caso estamos segmentando as imagens em regiões que pertencem a borda do objeto.

6.1. ACOMPANHAMENTO DE OBJETOS (VÍDEO)

A segmentação a partir de máscaras podem ser utilizados em vídeos digitais, sendo que os vídeos digitais são formados por frames, que por sua vez são imagens digitais. Nesta seção, será apresentado um algoritmo para segmentação por cor na faixa de azul em vídeos.

```
import cv2 as cv
import numpy as np
cap = cv.VideoCapture(0)
while(1):
    # Tome cada quadro
    _, frame = cap.read()
    # Converter BGR para HSV
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    # define o intervalo da cor azul no HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])
    # Limite a imagem do HSV para obter apenas cores azuis
    mask = cv.inRange(hsv, lower_blue, upper_blue)
    # Bitwise-AND máscara e imagem original
    res = cv.bitwise_and(frame,frame, mask= mask)
    cv.imshow('frame',frame)
    cv.imshow('mask',mask)
    cv.imshow('res',res)
    k = cv.waitKey(5) & 0xFF
    if k == 27:
        break
    cv.destroyAllWindows()
```

Exemplo de saída:

Fig. 29 – Representação de saída de Vídeo utilizando segmentação por cor.

6.2. LIMIARIZAÇÃO (THRESHOLDING)

Limiarização é uma abordagem para a segmentação fundamentada na análise da similaridade de níveis de cinza, de modo a extrair objetos de interesse mediante a definição de um limiar T que separa os agrupamentos de níveis de cinza da imagem. Uma das dificuldades do processo reside na determinação do valor mais adequado de limiarização, i.e., do ponto de separação dos pixels da imagem considerada. Através da análise do histograma da imagem, é possível estabelecer um valor para T na região do vale situado entre picos que caracterizam regiões de interesse na imagem. Há diversas variantes de limiarização. A mais simples delas é a técnica do particionamento do histograma da imagem por um limiar único T. A segmentação se dá varrendo-se a imagem, pixel a pixel, e rotulando-se cada pixel como sendo do objeto ou do fundo, em função da relação entre o valor do pixel e o valor do limiar. O sucesso deste método depende inteiramente de quão bem definidas estão as massas de pixels no histograma da imagem a ser segmentada.

6.3. SEGMENTAÇÃO ORIENTADA A REGIÕES

A segmentação orientada a regiões se fundamenta na similaridade dos níveis de cinza da imagem. O crescimento de regiões é um procedimento que agrupa pixels ou sub-regiões de uma imagem em regiões maiores. A variante mais simples da segmentação orientada a regiões é a agregação de pixels, que se fundamenta na definição de uma semente, i.e., um conjunto de pontos similares em valor de cinza, a partir do qual as regiões crescem com a agregação de cada pixel à semente à qual estes apresentam propriedades similares (e.g. nível de cinza, textura ou cor). A técnica apresenta algumas dificuldades fundamentais, se afigurando como problemas imediatos (i) a seleção de sementes que representem adequadamente as regiões de interesse; e (ii) a seleção de propriedades apropriadas para a inclusão de pontos nas diferentes regiões, durante o processo de crescimento. A disponibilidade da informação apropriada possibilita, em cada pixel, o cálculo do mesmo conjunto de propriedades que será usado para atribuir os pixels às diferentes regiões pré-definidas, durante o processo de crescimento. Caso o resultado de tal cálculo implique agrupamentos de valores das propriedades, os pixels cujas propriedades se localizarem mais perto do centróide desses agrupamentos poderão ser usados como sementes.

6.4. SEGMENTAÇÃO BASEADA EM BORDAS

A detecção de bordas, anteriormente discutida, possibilita a análise de descontinuidades nos níveis de cinza de uma imagem. As bordas na imagem de interesse caracterizam os contornos dos objetos nele presentes, sendo bastante úteis para a segmentação e identificação de objetos na cena. Pontos de borda podem ser

entendidos como as posições dos pixels com variações abruptas de níveis de cinza. Os pontos de borda caracterizam as transições entre objetos diferentes. Várias técnicas de segmentação baseiam-se na detecção de bordas, sendo as mais simples aquelas nas quais as bordas são detectadas pelos operadores de gradiente (e.g. Sobel, Roberts, Laplaciano), seguida de um processo de limiarização.

6.5. LIMIAR SIMPLES

Em Limiarização simples, se o valor do pixel for maior que um valor limite, ele é atribuído a um valor (pode ser branco), caso contrário, é atribuído outro valor (pode ser preto). A função usada é **cv.threshold**. O primeiro argumento é a imagem de origem, que deve ser uma imagem em tons de cinza . O segundo argumento é o valor limite usado para classificar os valores de pixel. O terceiro argumento é o maxVal que representa o valor a ser dado se o valor do pixel for maior que (às vezes menor que) o valor do limite. O OpenCV fornece diferentes estilos de limiar e é decidido pelo quarto parâmetro da função. Diferentes tipos são:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('gradient.png',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
   plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

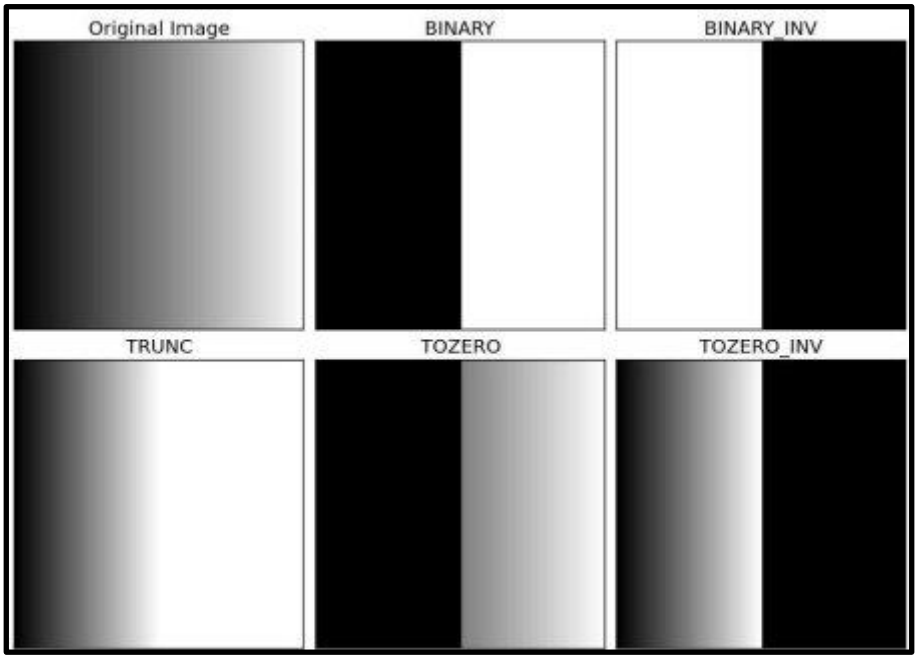

Fig. 30 – Representação de saída da utilização de segmentação limiar simples.

6.6. LIMIAR ADAPTATIVA

Na seção anterior, foi utilizado um valor global como valor limite. Mas pode não ser bom em todas as condições em que a imagem tem diferentes condições de iluminação em diferentes áreas. Nesse caso,o limiar adaptativo é melhor utilizado. O algoritmo calcula o limiar para pequenas regiões da imagem. Assim, obtemos limiares diferentes para diferentes regiões da mesma imagem e isso nos dá melhores resultados para imagens com iluminação variável.

Essa segmentação possui três parâmetros de entrada 'especiais' e apenas um argumento de saída.

Método Adaptativo - Decide como o valor de limiar é calculado.

- **cv.ADAPTIVE_THRESH_MEAN_C** : valor limite é a média da área da vizinhança.
- **cv.ADAPTIVE_THRESH_GAUSSIAN_C** : valor limite é a soma ponderada dos valores de vizinhança onde os pesos são uma janela gaussiana.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('sudoku.png',0)
img = cv.medianBlur(img,5)
ret,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,\
            cv.THRESH_BINARY,11,2)
th3 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv.THRESH_BINARY,11,2)
```

```
titles = ['Original Image', 'Global Thresholding (v = 127)',
          'Adaptive Mean Thresholding', 'Adaptive Gaussian
Thresholding']
images = [img, th1, th2, th3]
for i in xrange(4):
   plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

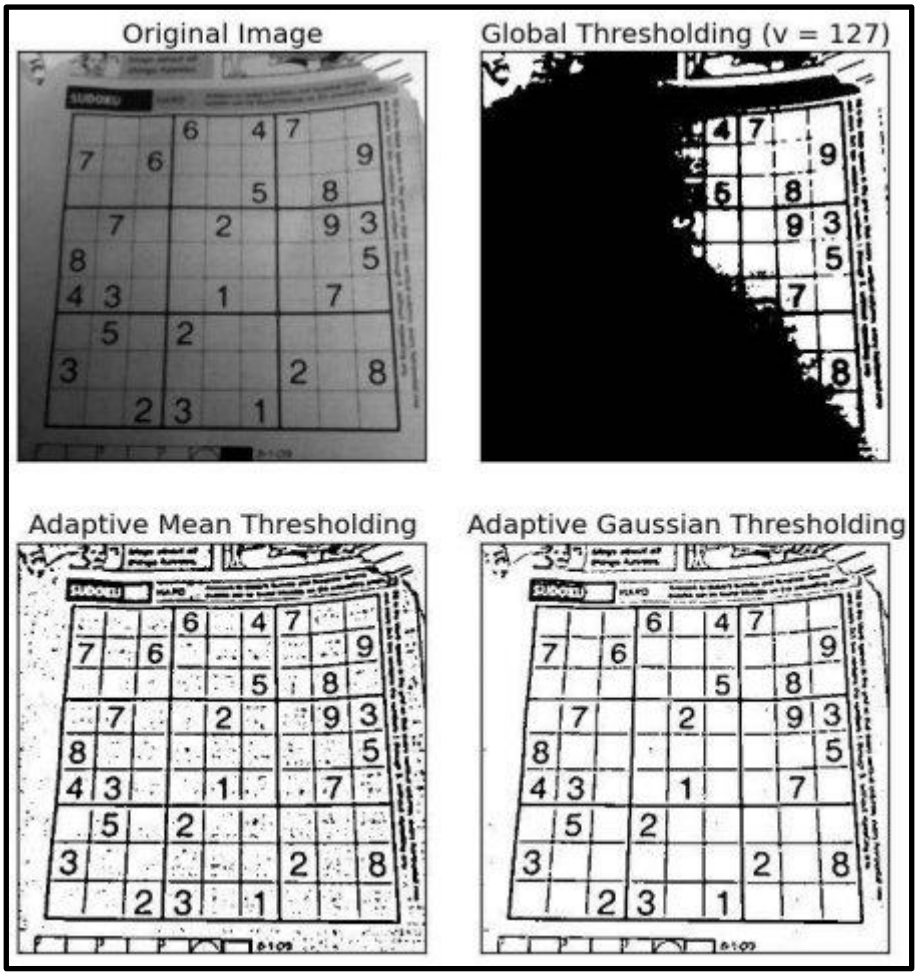

Fig. 31 – Representação de saída da utilização de segmentação limiar adaptativa.

6.7. BINARIZAÇÃO DE OTSU

No limiar global, foi utilizado um valor arbitrário para o valor limite, então, é difícil definir se esse valor limite selecionado é satisfatório ou não, considere uma imagem bimodal (*em palavras simples, imagem bimodal é uma imagem cujo histograma tem dois picos*). Para essa imagem, é possível obter aproximadamente um valor no meio desses picos como valor limite, isso é o que a binarização Otsu faz.

Então, em palavras simples, ele calcula automaticamente um valor limite do histograma da imagem para uma imagem bimodal. (Para imagens que não são bimodais, a binarização não será precisa).

Para isso, a função cv.threshold() é usada, mas passa um sinalizador extra, cv.THRESH_OTSU .

Confira abaixo o exemplo. A imagem de entrada é uma imagem ruidosa. No primeiro caso, foi aplicado o limiar global para um valor de 127. No segundo caso, foi aplicado o limiar de Otsu diretamente. No terceiro caso, acontece uma filtragem na imagem com um kernel gaussiano 5x5 para remover o ruído e, em seguida, aplicava o limiar Otsu. Veja como a filtragem de ruído melhora o resultado.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('noisy2.png',0)
# limiar global
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Limite de Otsu
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Limite de Otsu após carga gaussiana
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# traçar todas as imagens e seus histogramas
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding
(v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's
Thresholding"]
for i in range(3):
   plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
   plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
   plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
   plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
   plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
   plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
   plt.show()
```

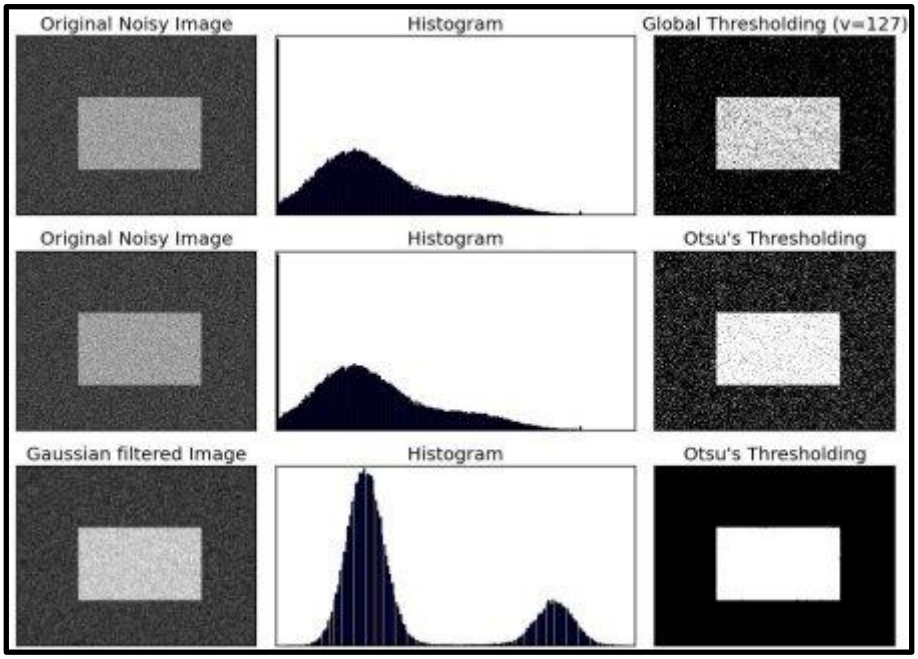

Fig. 32 – Representação do histograma e saída segmentada com otsu.

7. EXTRAÇÃO DE CARACTERÍSTICAS

Uma das tarefas mais complexas na análise de imagens está na definição de um conjunto de características que possam descrever de maneira concreta cada região contida em uma imagem, de modo a serem utilizados em processos de mais alto nível. Em outras palavras, a etapa de caracterização e representação da imagem consiste em uma etapa de fundamental importância.

Através de modelos/métodos matemáticos, as propriedades da imagem são representadas em formato de atributos, para então, servirem de entrada para uma posterior análise de reconhecimento e classificação de padrões. Em linhas gerais, a etapa de extração de características pode ser dividida em duas categorias de análises, sendo elas por: textura e forma. Em uma análise por textura, o objetivo geral é descrever aspectos da imagem no que diz respeito a suavidade, rugosidade e regularidade. Já em uma análise baseada na forma da imagem, o intuito é extrair informações que mensuram sobre propriedades morfológicas da imagem.

8. CLASSIFICAÇÃO E RECONHECIMENTO

Uma vez que os descritores da imagem dos objetos segmentados encontram-se disponíveis, passa-se à etapa seguinte, que consiste em distinguir objetos na imagem agrupando esses parâmetros de acordo com sua semelhança para cada região de pixels encontrada. Essa é a função dos processos de classificação e reconhecimento.

O processo de reconhecimento pode se dar em dois momentos em um sistema de visão computacional. No primeiro, as características são extraídas com o objetivo

de que os objetos sejam reconhecidos como pertencentes a um mesmo grupo e então sejam classificados em uma base de imagens. Em um segundo momento, novos objetos são apresentados ao sistema, que os reconhece, comparando suas características com aquelas dos objetos das classes previamente estabelecidas.

A partir da classificação dos objetos, considerando seus descritores, novos objetos podem ser reconhecidos, e é possível tomar decisões e relatar fatos relacionados aos objetos do mundo real. Existem diversas técnicas de classificação. As mais simples implicam processos de agrupamento estatístico, para os quais se necessita de alguma intervenção humana. As mais sofisticadas permitem ao computador reconhecer diferentes objetos através de técnicas com pouca ou nenhuma intervenção humana. Os processos que possuem intervenção humana são chamados de supervisionados.

A palavra classificação não denota nenhum juízo de valor, mas apenas o agrupamento em classes dos diversos objetos obtidos na segmentação. Em geral, vários atributos são necessários para uma correta classificação. Mas, quanto mais atributos, mais complexo se torna o problema. Dessa forma, é muito importante realizar uma seleção adequada dos atributos disponíveis.

9. EXERCÍCIOS

A. Construa um algoritmo que imite a função cv2.line da biblioteca OpenCV, aplique a função que foi desenvolvida em uma imagem e logo em seguida salve essa imagem.

B. Construa um algoritmo que imite a função cv2.rectangle da biblioteca OpenCV, aplique a função que foi desenvolvida em uma imagem e logo em seguida salve essa imagem.

C. Construa um algoritmo que imite a função cv2.circle da biblioteca OpenCV, aplique a função que foi desenvolvida em uma imagem e logo em seguida salve essa imagem.

D. Construa um algoritmo que imite o método size da biblioteca OpenCV, aplique a função que foi desenvolvida em uma imagem.

E. Construa um algoritmo que carregue uma imagem colorida, e logo em seguida separe os canais, e salve em imagens distintas cada canal.

F. Construa um algoritmo que calcule a moda, média e mediana dos valores de pixels de uma determinada imagem.

G. Construa um algoritmo que calcule a variância dos valores de pixel dada uma determinada imagem.

H. Construa um algoritmo que corte uma imagem, as entradas da função são as coordenadas (X,Y)

I. Construa um algoritmo que dada duas imagens, o algoritmo faça a adição dos valores dos pixels das duas imagens, salve a nova imagem.

J. Construa um algoritmo que utilize em uma mesma imagem as técnicas de Erosão, Abertura, Gradiente Morfológico,Black Hat.

K. Construa um algoritmo que utilize em uma mesma imagem as técnicas de Dilatação, Abertura, Gradiente Morfológico e Black Hat.

L. Construa um algoritmo que equalize um histograma, sem a utilização da função cv2.equalizeHist.

M. Utilize o algoritmo do tópico 6.1 e mude a cor onde a segmentação irá acontecer.