Inteligência Artificial

Instrutores
Ph.D. Professor Aluisio Igor Rego Fontes

Segmentação

Segmentação

- O que é segmentação?

Segmentação

- O que é segmentação?
  - A segmentação em processamento de imagem refere-se à técnica de dividir uma imagem em partes ou regiões distintas, com base em características como cor, intensidade, textura ou bordas.

Como segmentar?

Na segmentação de imagens, há vários critérios de segmentação diferentes para segmentar uma imagem.

Vamos ver em específico três tipos de segmentação:

- Segmentação por limiarização
- Segmentação por crescimento de regiões
- Segmentação por baseada em bordas

Limiarização (Thresholding)

- O que é limiarização?
  - Também conhecida como binarização
  - Se um pixel tem um valor menor que o limiar, ele é atribuído a uma classe ou valor específico; se for maior ou igual ao limiar, é atribuído à outra classe ou valor.
  - “Traçar a reta”.

Algoritmos para limiarização

- Algoritmo de Otsu
  - Algoritmo utilizado para tentar encontrar o limiar ótimo para a limiarização, em vez de escolher um limite fixo, o algoritmo que escolhe qual vai ser esse limiar
  - A ideia central do algoritmo é encontrar o limiar que minimiza a variabilidade intraclasse e maximiza a variabilidade entre as classes.
    - Ou seja, tem o objetivo de encontrar o limiar que deixe classes distintas o mais longe possível e os objetos (pixels) pertencentes às classes o mais próximo possível.

Limiarização em Python

- Código para fazer limiarização manual e limiarização Otsu utilizando **OpenCV**
- Vamos utilizar de exemplo essa imagem:

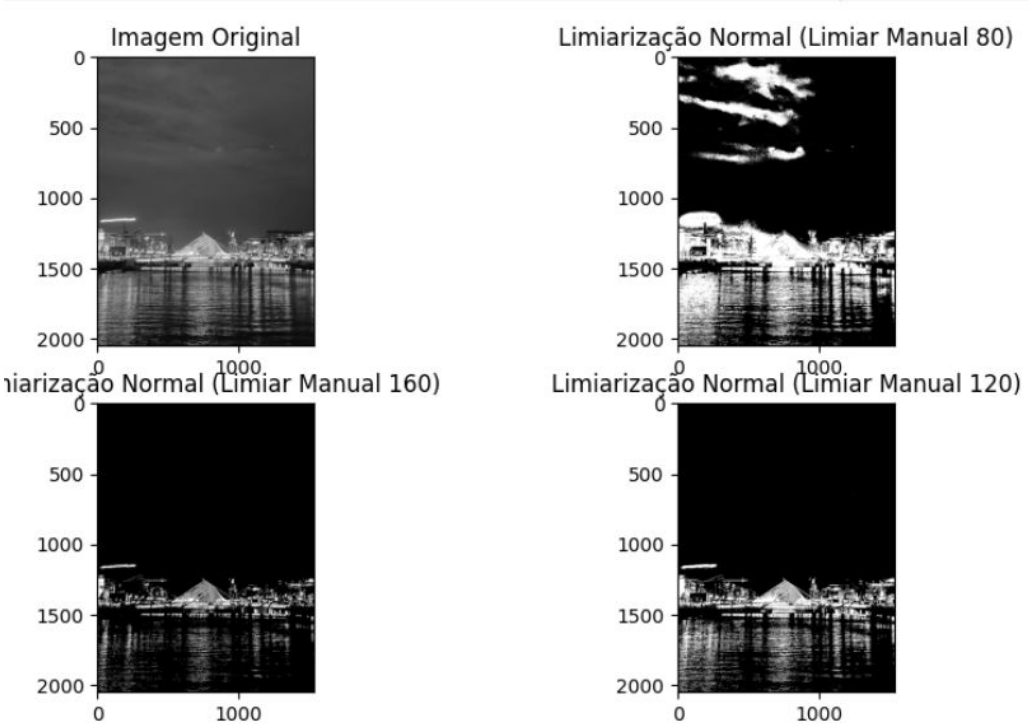
Limiarização em Python

- Para fazer a binarização é importante que a imagem esteja em escala de cinza, pois o algoritmo irá dividir em duas classes (branco e preto)

Código para a binarização manual em python

```
imagem_path = 'imagem2.jpg'
imagem = cv2.imread(imagem_path, cv2.IMREAD_GRAYSCALE)

limiar_manual1 = 80
_, binarizada_normal1 = cv2.threshold(imagem, limiar_manual1, 255, cv2.THRESH_BINARY)
```

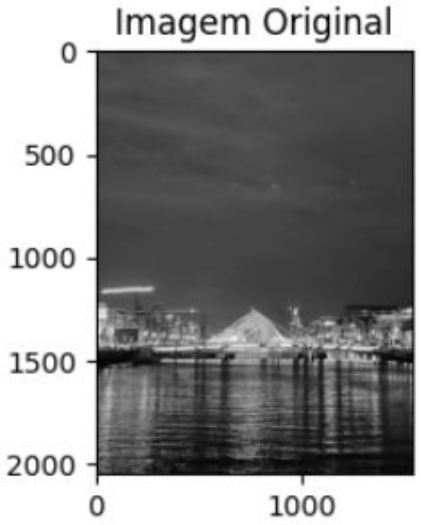
Código para a binarização manual em python

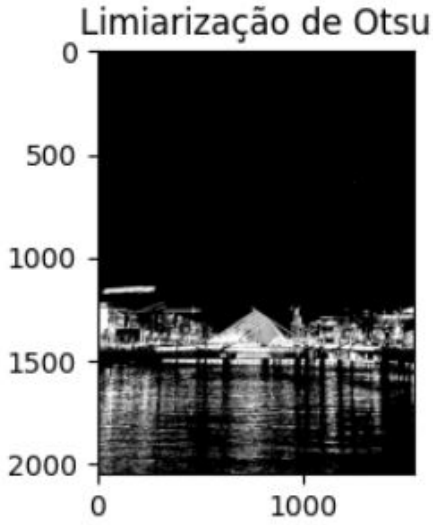


Código para a binarização Otsu em python

```
imagem_path = 'imagem2.jpg'
imagem = cv2.imread(imagem_path, cv2.IMREAD_GRAYSCALE)

_, binarizada_otsu = cv2.threshold(imagem, 0, 255, cv2.THRESH_OTSU)
```

Código para a binarização Otsu em python





O que é segmentação baseada em Bordas?

- A segmentação baseada em bordas é uma técnica de processamento de imagem que visa identificar e realçar bordas ou transições significativas entre diferentes regiões na imagem. As bordas são regiões onde ocorre uma mudança abrupta nas propriedades da imagem, como intensidade de cor, textura ou luminosidade.

O que é segmentação baseada em Bordas?

- As duas técnicas mais comuns para detecção de bordas são: Algoritmos de detecção de bordas Canny e Algoritmo Sobel.

Algoritmos para segmentação baseada em bordas

- Canny
  - O Canny é um método popular para detectar bordas em imagens. Ele utiliza um filtro gaussiano para suavizar a imagem, seguido pela aplicação de operadores de gradiente para identificar mudanças abruptas na intensidade. Finalmente, é aplicado um algoritmo de não-máxima supressão e histerese para obter bordas finais.
- Sobel
  - O Algoritmo de Sobel é um operador de gradiente utilizado para realçar bordas em imagens. Ele calcula as derivadas parciais da imagem em relação às coordenadas x e y, combinando-as para calcular a magnitude do gradiente. Isso resulta em uma imagem onde as bordas são mais proeminentes.

Canny Edge e Sobel em Python

- Código do Canny em Python

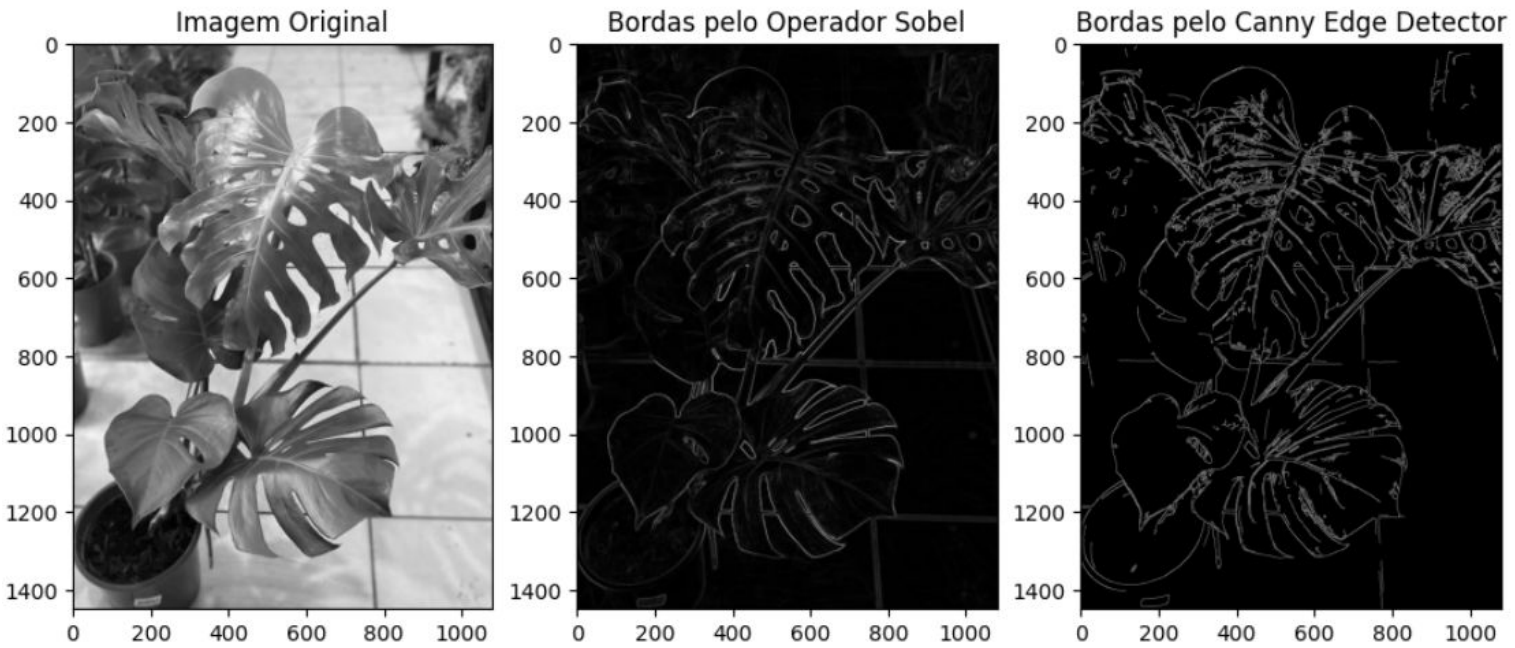
```
imagem_path = 'costela_adao.jpg'
imagem = cv2.imread(imagem_path, cv2.IMREAD_GRAYSCALE)

canny_edges = cv2.Canny(imagem, 50, 150)
```

Canny Edge e Sobel em Python

- Código do Sobel em Python

```
imagem_path = 'costela_adao.jpg'
imagem = cv2.imread(imagem_path, cv2.IMREAD_GRAYSCALE)

sobel_x = cv2.Sobel(imagem, cv2.CV_64F, 1, 0, ksize=3) # Derivação em relação ao X
sobel_y = cv2.Sobel(imagem, cv2.CV_64F, 0, 1, ksize=3) # Derivação em relação ao Y
sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2) # Magnitude do gradiente
```

Canny Edge e Sobel em Python



Avaliação da segmentação

- Como sei que uma segmentação foi boa ou não?
- Primeiro, vamos definir o que é uma boa segmentação

Avaliação da segmentação

- Para ilustrar, vamos imaginar que foi lhe passado uma tarefa para identificação de contorno de várias células em uma imagem microscópica
- Boa segmentação:
  - Uma boa segmentação seria aquela em que os contornos identificados correspondem de perto aos limites reais das células. As regiões segmentadas deveriam ser bem definidas e corresponder às bordas corretas
- Má segmentação:
  - Em contrapartida, uma má segmentação pode ocorrer se os contornos das células não forem identificados corretamente, resultando em regiões sobrepostas ou segmentos imprecisos. Isso pode ocorrer devido a métodos inadequados, ruído na imagem ou escolha inadequada de parâmetros.

Avaliação da segmentação

- Como faço para avaliar uma segmentação?
  - Há métricas que podem nos ajudar
- *Precision* (Precisão)
  - Mede a proporção de pixels corretamente classificados como pertencentes à classe alvo em relação a todos os pixels classificados como pertencentes a essa classe.
- *Recall*
  - Mede a proporção de pixels corretamente classificados como pertencentes à classe alvo em relação a todos os pixels que realmente pertencem a essa classe.
- Índice de Jaccard (*IoU - Intersection over Union*)
  - Mede a sobreposição entre a região segmentada e a verdadeira região. Quanto maior o IoU, melhor a sobreposição

Avaliação da segmentação

- Código em python para avaliação de métricas
- Nesse exemplo, vamos fazer um exemplo ilustrativo onde em uma imagem contém os contornos verdadeiros da imagem e uma imagem de uma segmentação feita.
- Para começo, precisamos importar as métricas da biblioteca sklearn.

```
from sklearn.metrics import precision_score, recall_score, jaccard_score
```

Avaliação da segmentação

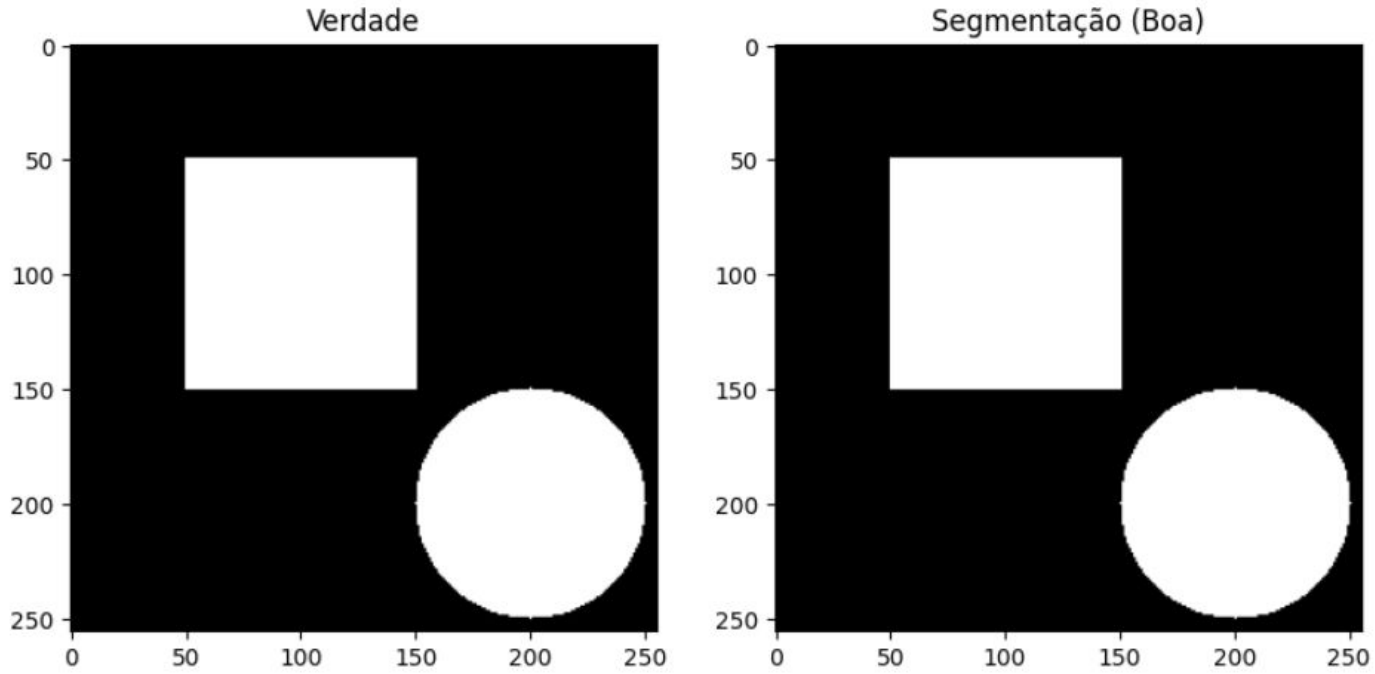
- Função feita para resumir as três métricas

```python
def evaluate_segmentation(true_image, segmentation):
    # Mapear os valores 255 para 1
    true_image = true_image // 255
    segmentation = segmentation // 255

    precision = precision_score(true_image.flatten(), segmentation.flatten(), average='binary')
    recall = recall_score(true_image.flatten(), segmentation.flatten(), average='binary')
    jaccard = jaccard_score(true_image.flatten(), segmentation.flatten(), average='binary')

    print(f'Precisão: {precision:.2f}')
    print(f'Recall: {recall:.2f}')
    print(f'Índice de Jaccard: {jaccard:.2f}')

    return precision, recall, jaccard
```

Avaliação da segmentação

- Como interpretar as métricas?
  - Essas métricas variam de 0 a 1, quanto mais próximo de 1 estiver, melhor a segmentação feita, quanto mais próximo de 0 pior a segmentação.

Avaliação da segmentação



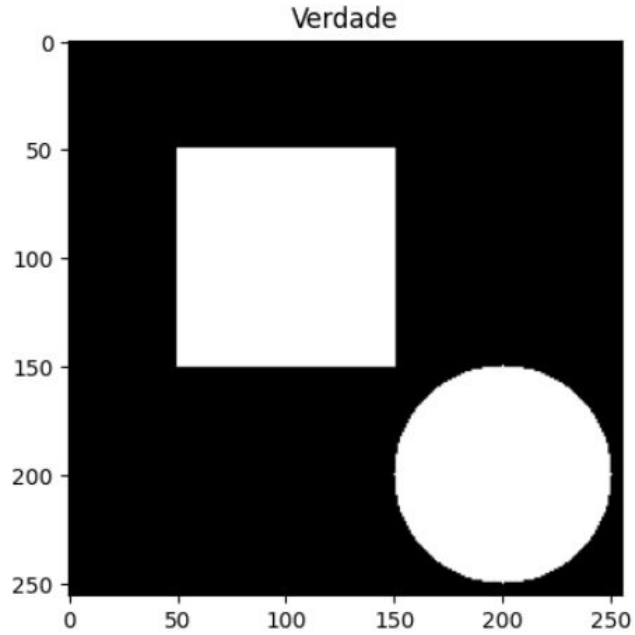
Precisão: 1.00
Recall: 1.00
Índice de Jaccard: 1.00

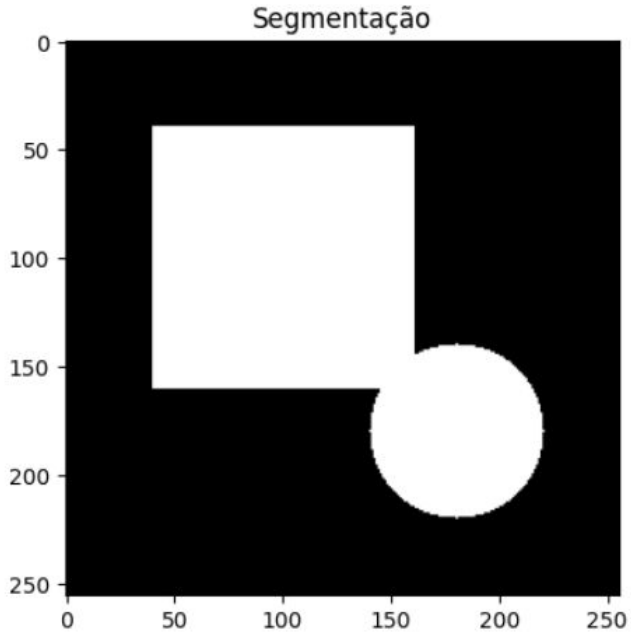
Avaliação da segmentação

```
Precisão: 1.00
Recall: 1.00
Índice de Jaccard: 1.00
```

- Nesse exemplo de boa segmentação, podemos ver que os 3 índices deram perfeitos, ou seja, temos uma segmentação perfeita

Avaliação da segmentação





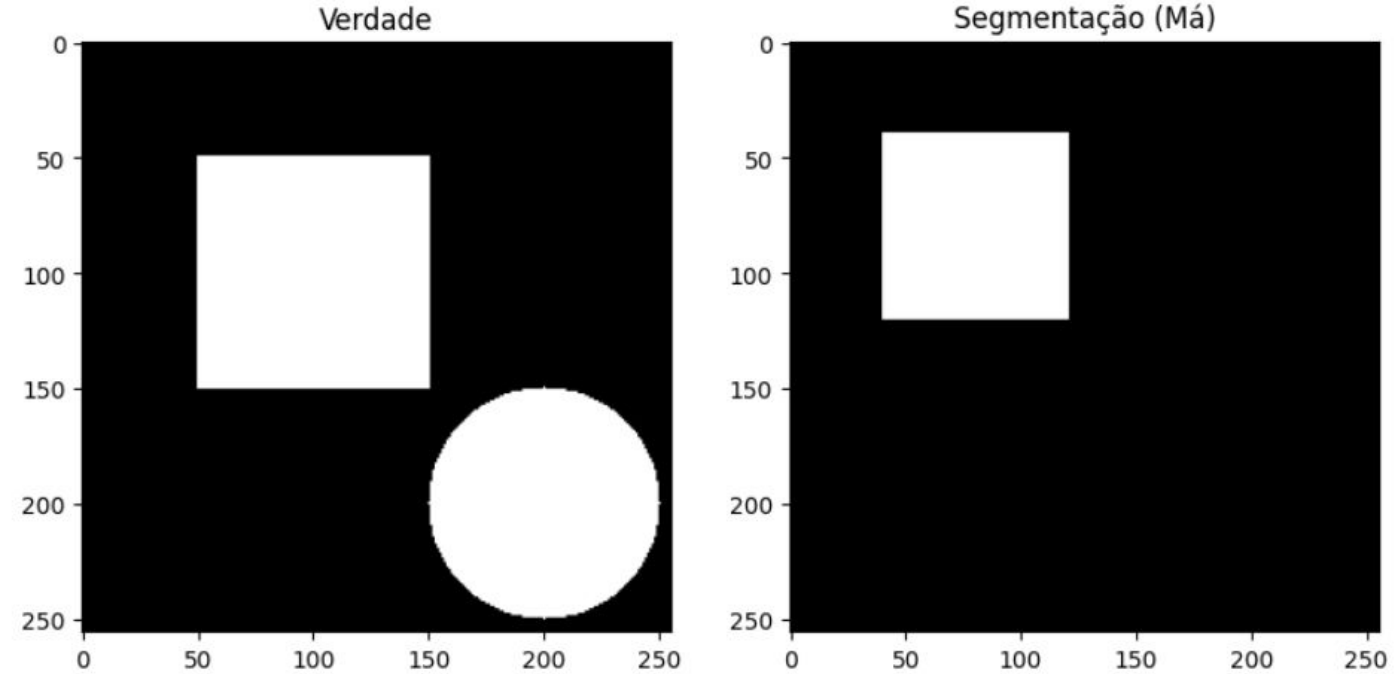
Precisão: 0.72
Recall: 0.77
Índice de Jaccard: 0.59

Avaliação da segmentação

```
Precisão: 0.72
Recall: 0.77
Índice de Jaccard: 0.59
```

- Neste outro exemplo, temos uma segmentação mediana-ruim, dois índices (*precision e recall)* deram acima de 0.7 e o jaccard deu entre 0.5 e 0.6, o que nos diz que há algumas partes sobrepostas que prejudicam a segmentação.

Avaliação da segmentação



Precisão: 0.77
Recall: 0.28
Índice de Jaccard: 0.26

Avaliação da segmentação

```
Precisão: 0.77
Recall: 0.28
Índice de Jaccard: 0.26
```

- Aqui, temos um exemplo de uma segmentação ruim, por mais que a precisão tenha dado razoável, as outras métricas deram péssimas (abaixo de 0.5).

Reconhecimento de Padrões

Reconhecimento de Padrões

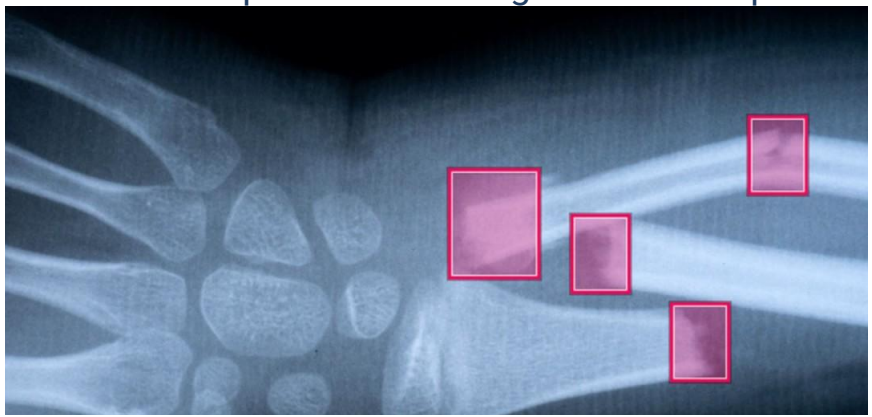
Reconhecimento de Padrões

- Definição: O reconhecimento de padrões é o processo de identificar padrões regulares em dados usando técnicas de aprendizado de máquina e visão computacional.
  - Em termos simples, é como ensinar uma máquina a 'ver' e 'entender' padrões, como identificar rostos, objetos, ou tendências em dados.

- Importância: Esta área é crucial porque permite que computadores interpretem dados complexos, tornando possível a automação e análises sofisticadas que seriam impossíveis ou extremamente demoradas para os seres humanos.

Reconhecimento de Padrões

- Aplicações no Mundo Real:
  - Reconhecimento Facial: Usado em smartphones para desbloqueio e em sistemas de segurança.
  - Leitura Automática de Placas: Ajuda no monitoramento do tráfego e na gestão de estacionamentos.
  - Diagnósticos Médicos: Identifica padrões em imagens médicas para auxiliar diagnósticos.

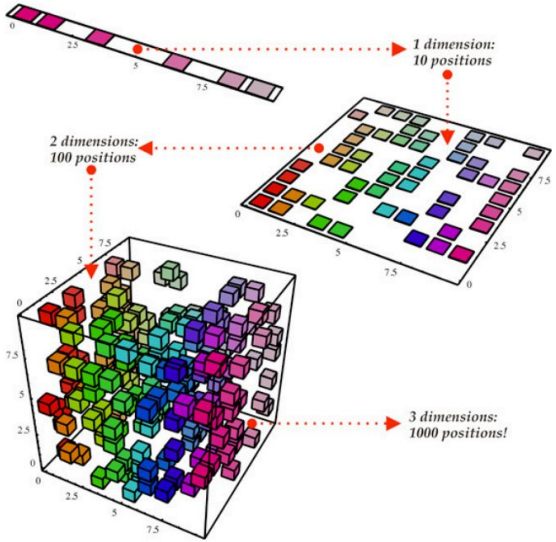


Métodos Estatísticos

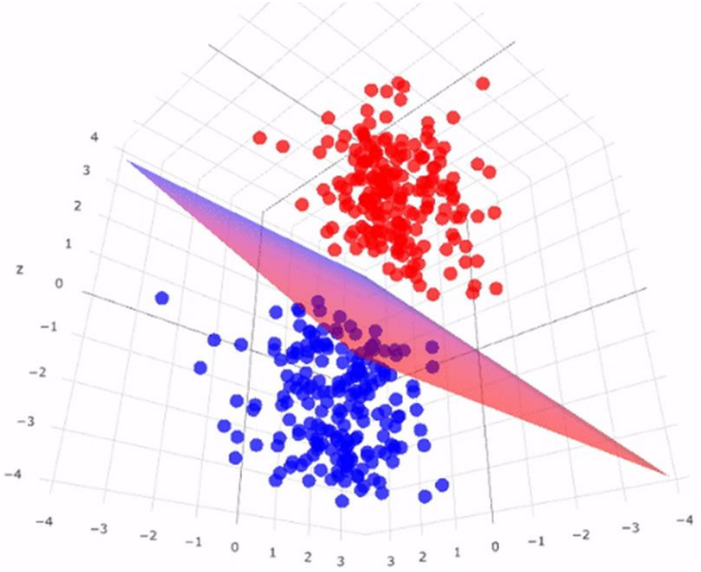
- Estes métodos envolvem analisar e interpretar conjuntos de dados para encontrar padrões e relações. Usam a matemática e a probabilidade para fazer previsões ou classificações baseadas em dados.
  - PCA (Análise de Componentes Principais): Técnica usada para simplificar um conjunto de dados, reduzindo-o a suas componentes mais importantes, o que facilita a análise sem perder muitas informações relevantes.
  - SVM (Máquinas de Vetores de Suporte): Um modelo de aprendizado de máquina usado para classificar dados. Ele encontra o melhor 'limite' que separa diferentes categorias de dados.

Métodos Estatísticos

PCA



SVM



Métodos Estatísticos

```
# Criar um pipeline com PCA e SVM
clf = make_pipeline(StandardScaler(), PCA(n_components=30), SVC(gamma='auto'))
clf.fit(X_train, y_train)
```

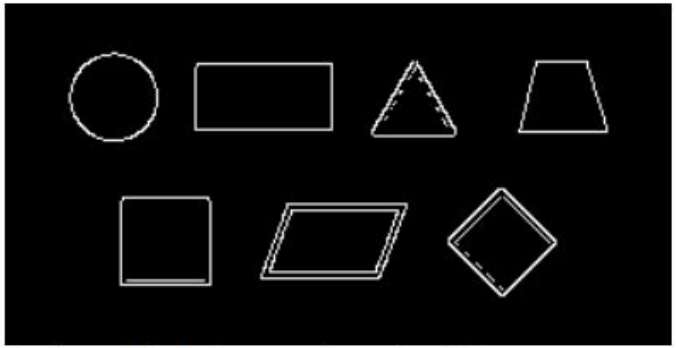
Métodos Estruturais

- Focam na estrutura interna dos dados.
  - Por exemplo, na análise de imagens, esses métodos podem identificar e classificar formas e objetos com base em suas características estruturais, como bordas e contornos.

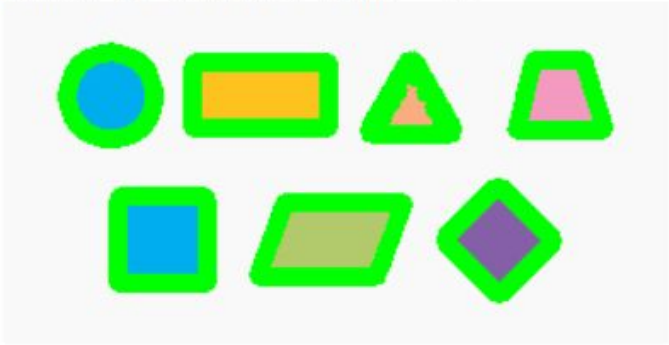
- Detecção de Bordas
  - Usada para identificar os limites de objetos em uma imagem.

- Reconhecimento de Formas
  - Identifica formas geométricas específicas, como círculos ou triângulos.

Métodos Estruturais



Number of Contours found = 12



Métodos Estruturais

```
# Aplicar detecção de bordas de Canny
edged = cv2.Canny(gray, 30, 200)
```

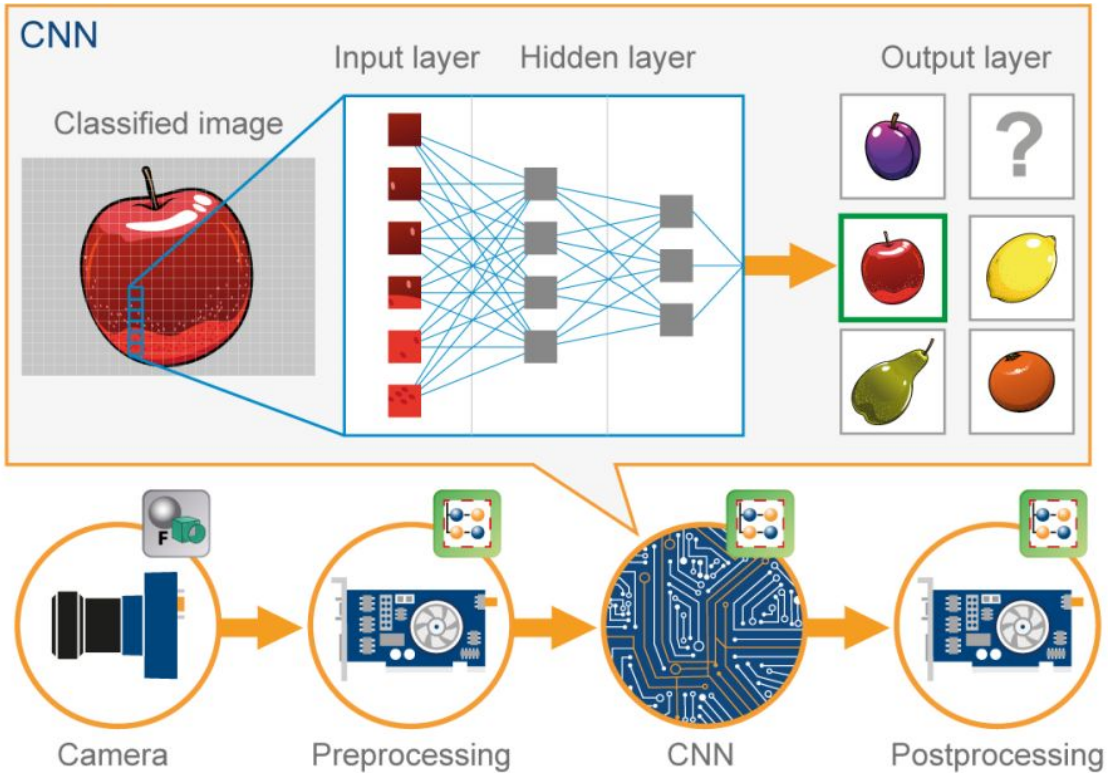
```
# Encontrar contornos
contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
for contour in contours:
    # Aproximar os contornos para formas
    approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
    cv2.drawContours(image, [approx], 0, (0, 0, 255), 5)
    if len(approx) == 3:
        shape_name = "Triângulo"
    elif len(approx) == 4:
        shape_name = "Quadrado ou Retângulo"
    elif len(approx) > 10:
        shape_name = "Círculo"
    else:
        shape_name = "Polígono"

    # Adicionar nome da forma na imagem
    cv2.putText(image, shape_name, (contour[0][0][0], contour[0][0][1]), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0), 2)
```

Redes Neurais Convolucionais (CNNs)

- As **CNNs** são um tipo especializado de rede neural projetado para processar dados que têm uma estrutura de grade, como imagens.
  - Uma imagem, por exemplo, é basicamente uma grade de pixels. O ponto chave das **CNNs** é que elas automatizam a detecção de características importantes.
    - Sem necessidade de intervenção humana.
- Pontos em destaque:
  - Camadas em CNNs
  - Funcionamento e Aplicações
  - Por que são eficazes?

Redes Neurais Convolucionais (CNNs)



Redes Neurais Convolucionais (CNNs)

- Camadas em CNNs:
  - Camadas Convolutivas:
    - São o coração das CNNs. Eles aplicam uma operação chamada "convolução". Imagine um pequeno filtro deslizando sobre toda a imagem. Este filtro é focado em extrair características como bordas, texturas, ou padrões específicos.

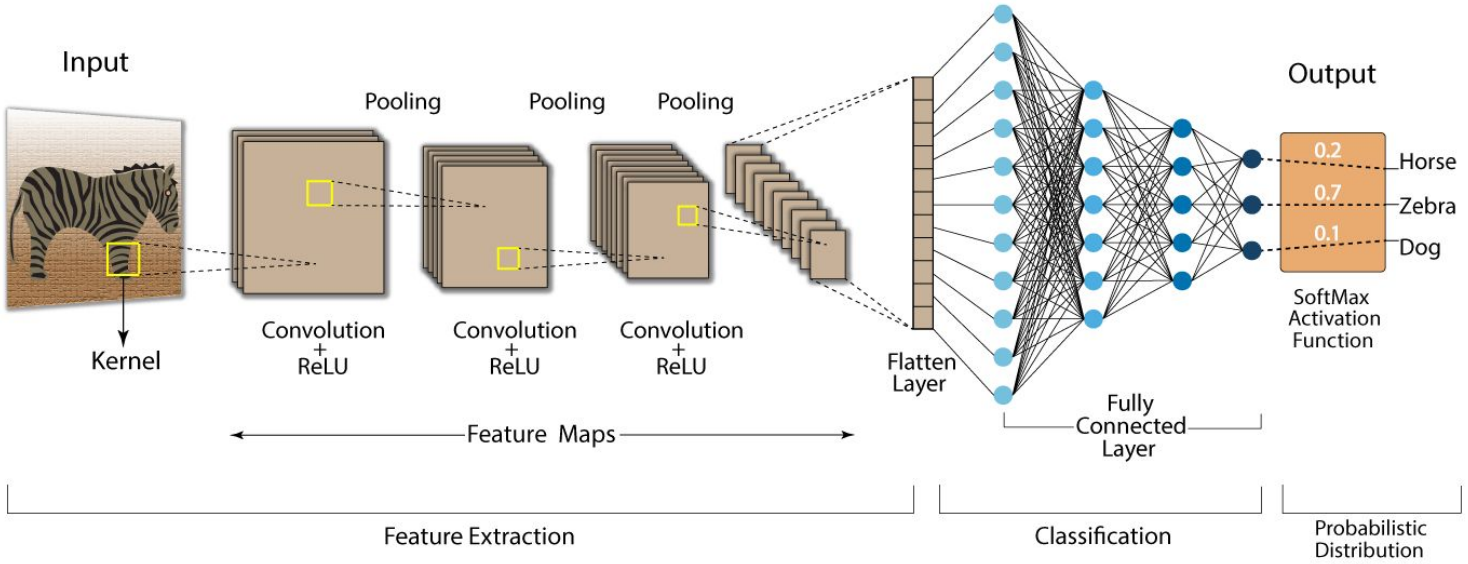
  - Camadas de Pooling (Agrupamento):
    - Após as camadas convolutivas, geralmente vêm as camadas de pooling. Elas ajudam a reduzir a dimensão espacial (altura e largura, mas não profundidade) dos dados de entrada. Isso é feito para diminuir o processamento computacional e evitar overfitting. O tipo mais comum de pooling é o max pooling, que simplesmente pega o valor máximo em uma janela definida.
  -

Redes Neurais Convolucionais (CNNs)

- Camadas em CNNs:
  - Camadas Fully Connected (Totalmente Conectadas):
    - No final de uma CNN, temos camadas totalmente conectadas.
    - Aqui, os dados que foram aprendidos e condensados pelas camadas convolutivas e de pooling são usados para classificar a imagem em uma categoria (como identificar se uma imagem é de um gato ou um cachorro).

Redes Neurais Convolucionais (CNNs)

- Camadas em CNNs



Redes Neurais Convolucionais (CNNs)

- Funcionamento e Aplicações:
  - As CNNs aprendem a reconhecer padrões visuais, começando por padrões muito simples:
    - Bordas, em suas primeiras camadas, e progredindo para conceitos mais complexos e abstratos em camadas mais profundas.

  - Elas são amplamente utilizadas em várias aplicações, como reconhecimento de imagens, diagnósticos médicos por imagem, sistemas de vigilância, e até mesmo na identificação de padrões em jogos.

Redes Neurais Convolucionais (CNNs)

- Por que são eficazes?
  - Preservação da Informação Espacial:
    - Ao contrário de outras redes neurais, as CNNs preservam as relações espaciais entre pixels, o que é crucial para entender imagens.
  - Redução de Parâmetros:
    - Devido ao compartilhamento de pesos e ao uso de filtros menores, as CNNs têm menos parâmetros do que uma rede totalmente conectada de tamanho comparável, tornando-as mais eficientes.

Redes Neurais Convolucionais (CNNs)

- Imagine as **CNNs** como uma linha de montagem em uma fábrica, onde cada etapa (camada) tem uma tarefa especializada.
  - No início, são identificadas características simples, e à medida que o produto passa por diferentes estágios (camadas), características mais complexas são montadas a partir desses componentes simples.
  - No final da linha, temos um produto acabado (uma decisão de classificação ou reconhecimento).

Redes Neurais Convolucionais (CNNs)

- Exemplo no notebook:
  - Reconhecimento_de_Padrões.ipynb
  - Tem uma imagem que acompanha e deve ser colocada na raiz do Notebook para que funcione
    - geometric_shapes.jpg


OBRIGADO!