Resumo: Pré-processamento, Segmentação e Detecção/Classificação de Imagens

1. Pré-processamento de Imagens

Introdução

O pré-processamento de imagens é uma etapa essencial que prepara os dados visuais para análises mais complexas. Basicamente, é onde a gente "limpa" e melhora a qualidade das imagens antes de aplicar algoritmos de machine learning e/ou visão computacional.

Essa fase envolve várias técnicas que ajudam a reduzir ruídos, normalizar iluminação, ajustar contraste e converter formatos. O objetivo principal é deixar a imagem em condições ideais para que os algoritmos posteriores funcionem melhor. É como preparar os ingredientes antes de cozinhar (sem essa preparação adequada, o resultado final pode não ser tão satisfatório).

As principais operações incluem redimensionamento, filtragem, correção de cores, remoção de ruído e normalização. Cada uma dessas técnicas tem sua importância dependendo do tipo de problema que estamos tentando resolver.

Exemplos de bibliotecas/frameworks

OpenCV é disparado a biblioteca mais popular para processamento de imagens. Ela oferece praticamente tudo que você precisa, desde operações básicas até técnicas avançadas. É multiplataforma e tem suporte para Python, C++ e outras linguagens.

PIL/Pillow é outra opção muito usada, especialmente para operações mais simples. É mais leve que o OpenCV e integra bem com o ecossistema Python. Ideal para redimensionamento, rotação e conversões básicas.

scikit-image faz parte do scikit-learn e é ótima para quem já trabalha com esse ecossistema. Tem implementações bem otimizadas de vários algoritmos clássicos de processamento de imagem.

ImageIO é útil para leitura e escrita de diferentes formatos de imagem. Funciona bem em conjunto com outras bibliotecas.

Exemplos de aplicações

Aqui está um exemplo prático de pré-processamento usando OpenCV:

python

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Carrega a imagem
img = cv2.imread('imagem_exemplo.jpg')

# Converte de BGR para RGB (OpenCV usa BGR por padrão)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Redimensiona a imagem para um tamanho padrão
img_resized = cv2.resize(img_rgb, (224, 224))

# Aplica filtro gaussiano para reduzir ruído
img_filtered = cv2.GaussianBlur(img_resized, (5, 5), 0)

# Normaliza os valores dos pixels (0-1)
img_normalized = img_filtered.astype('float32') / 255.0

# Ajusta o contraste usando equalização de histograma
img_gray = cv2.cvtColor(img_filtered, cv2.COLOR_RGB2GRAY)
img_equalized = cv2.equalizeHist(img_gray)
```

Este código mostra as operações mais comuns: redimensionamento (importante para padronizar entradas), filtragem gaussiana (remove ruídos), normalização (deixa os valores em escala adequada) e equalização de histograma (melhora o contraste).

Outro exemplo usando Pillow para operações mais simples:

python

```python
from PIL import Image, ImageEnhance
import numpy as np

# Abre a imagem
img = Image.open('foto.jpg')

# Redimensiona mantendo proporção
img = img.resize((300, 300), Image.Resampling.LANCZOS)

# Ajusta brilho e contraste
enhancer = ImageEnhance.Brightness(img)
img = enhancer.enhance(1.2)  # Aumenta brilho em 20%

enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(1.1)  # Aumenta contraste em 10%
```

```
# Converte para array numpy para processamento posterior
img_array = np.array(img)
```

---

2. Segmentação de Imagens

Introdução

A segmentação de imagens é o processo de dividir uma imagem em regiões ou segmentos distintos, onde cada segmento representa algo específico: pode ser um objeto, uma pessoa, o fundo, etc. É como se a gente estivesse "recortando" digitalmente diferentes partes da imagem.

Existem várias abordagens para segmentação: algumas baseadas em similaridade de cores (como K-means), outras em detecção de bordas (como Watershed), e métodos mais modernos usando deep learning (como U-Net e Mask R-CNN). A escolha da técnica depende muito do tipo de imagem e do nível de precisão necessário.

A segmentação é crucial em muitas aplicações: diagnóstico médico (segmentar tumores em exames), carros autônomos (identificar pedestres, outros carros, sinalizações), agricultura de precisão (identificar plantas doentes), entre outras.

Exemplos de bibliotecas/frameworks

OpenCV também é forte na segmentação, oferecendo algoritmos clássicos como Watershed, GrabCut e detecção de contornos. É uma boa escolha para começar e para casos onde métodos tradicionais são suficientes.

scikit-image tem implementações excelentes de algoritmos como SLIC (superpixels), Felzenszwalb, e Quick Shift. É ideal para pesquisa e prototipagem rápida.

TensorFlow/Keras e **PyTorch** são essenciais quando você quer usar deep learning para segmentação. Permitem implementar redes como U-Net, DeepLab, e outras arquiteturas modernas.

Segment Anything Model (SAM) da Meta é uma ferramenta mais recente que permite segmentação com prompts mínimos. Muito poderosa para casos gerais.

Exemplos de aplicações

Exemplo de segmentação usando K-means (método clássico):

```
python
import cv2
import numpy as np
from sklearn.cluster import KMeans
```

```
# Carrega e prepara a imagem
img = cv2.imread('paisagem.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Reshape da imagem para lista de pixels
data = img.reshape((-1, 3))
data = np.float32(data)

# Aplica K-means com 4 clusters (4 segmentos)
k = 4
kmeans = KMeans(n_clusters=k, random_state=42)
labels = kmeans.fit_predict(data)

# Reconstrói a imagem segmentada
centers = kmeans.cluster_centers_
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

segmented_img = segmented_data.reshape(img.shape)
```

Esse código agrupa pixels similares em cores, criando uma versão simplificada da imagem com apenas 4 cores principais. Útil para separar grandes regiões como céu, terra, vegetação, etc…

Exemplo mais avançado usando Watershed:

python

```
import cv2
import numpy as np
from scipy import ndimage
from skimage.segmentation import watershed
from skimage.feature import peak_local_maxima

# Carrega imagem em escala de cinza
img = cv2.imread('objetos.jpg', 0)

# Aplica threshold para binarizar
_, binary = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Calcula distância transform
dist_transform = cv2.distanceTransform(binary, cv2.DIST_L2, 5)

# Encontra picos locais (centros dos objetos)
local_peaks = peak_local_maxima(dist_transform, min_distance=20)

# Cria marcadores para watershed
markers = np.zeros_like(img)
for i, peak in enumerate(local_peaks[0]):
```

```
    markers[peak[0], peak[1]] = i + 1

# Aplica watershed
labels = watershed(-dist_transform, markers, mask=binary)
```

O Watershed é excelente para separar objetos que estão tocando uns nos outros, como células em microscopia ou frutas em uma cesta.

---

3. Detecção/Classificação de Imagens

Introdução

Detecção e classificação são duas tarefas relacionadas, porém distintas em visão computacional. A classificação responde "o que é isso?", ou seja, identifica qual categoria ou classe um objeto pertence. Já a detecção vai além, respondendo "o que é e onde está?", ou seja, não só identifica o objeto como também localiza ele na imagem.

A classificação tradicionalmente usava features engineered (como HOG, SIFT) combinadas com classificadores como SVM. Hoje em dia, redes neurais convolucionais (CNNs) dominam completamente essa área, com arquiteturas como ResNet, EfficientNet e Vision Transformers alcançando precisões impressionantes.

Para detecção, temos duas abordagens principais: métodos em dois estágios (como R-CNN) que primeiro propõem regiões e depois classificam, e métodos em um estágio (como YOLO e SSD) que fazem tudo de uma vez, sendo mais rápidos mas às vezes menos precisos.

Exemplos de bibliotecas/frameworks

TensorFlow/Keras é provavelmente a escolha mais popular. Tem modelos pré-treinados através do TensorFlow Hub, suporte excelente para transfer learning, e integração com TensorFlow Lite para mobile.

PyTorch é muito usado na pesquisa e está ganhando terreno na produção também. O torchvision tem muitos modelos prontos e é mais flexível para experimentação.

OpenCV DNN permite carregar modelos treinados em outros frameworks. É útil quando você quer apenas inferência sem toda a complexidade de treinar modelos.

Ultralytics YOLO é uma implementação moderna e fácil de usar da família YOLO. Excelente para detecção de objetos em tempo real.

Detectron2 do Facebook. Mais complexo de usar, mas resultados excelentes.

Exemplos de aplicações

Exemplo de classificação usando um modelo pré-treinado:

python
```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

# Carrega modelo pré-treinado
model = ResNet50(weights='imagenet')

# Carrega e prepara a imagem
img_path = 'gato.jpg'
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Adiciona dimensão batch
img_array = preprocess_input(img_array)

# Faz a predição
predictions = model.predict(img_array)
results = decode_predictions(predictions, top=3)[0]

# Mostra os resultados
for i, (imagenet_id, label, score) in enumerate(results):
    print(f"{i+1}. {label}: {score:.4f}")
```

Este código usa uma ResNet50 pré-treinada na ImageNet para classificar qualquer imagem em uma das 1000 categorias possíveis. O preprocess_input normaliza a imagem do jeito que o modelo espera.

Exemplo de detecção usando YOLO:

python
```python
import cv2
from ultralytics import YOLO

# Carrega o modelo YOLO pré-treinado
model = YOLO('yolov8n.pt')

# Carrega a imagem
img = cv2.imread('rua.jpg')

# Faz a detecção
results = model(img)

# Desenha as bounding boxes
for result in results:
```

```
    boxes = result.boxes
    for box in boxes:
        # Extrai coordenadas e confiança
        x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
        conf = box.conf[0].cpu().numpy()
        cls = box.cls[0].cpu().numpy()

        # Desenha retângulo se confiança > 0.5
        if conf > 0.5:
            cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 2)
            label = f'{model.names[int(cls)]}: {conf:.2f}'
            cv2.putText(img, label, (int(x1), int(y1)-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# Salva resultado
cv2.imwrite('resultado_detecção.jpg', img)
```

O YOLO é fantástico para detectar múltiplos objetos em tempo real. Este código detecta pessoas, carros, animais e dezenas de outras classes, desenhando caixas ao redor de cada detecção.

Exemplo de classificação customizada (transfer learning):

python

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Base pré-treinada (sem a camada final)
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congela as camadas da base
base_model.trainable = False

# Adiciona camadas customizadas
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)  # 3 classes customizadas

# Cria o modelo final
model = Model(inputs=base_model.input, outputs=predictions)

# Compila o modelo
model.compile(optimizer='adam',
```

```
loss='categorical_crossentropy',
metrics=['accuracy'])
```