Resumo Módulo 3

Andressa Colares

Agosto 2024

1 O que é pré-processamento de imagens?

Em projetos de visão computacional, o pré-processamento de imagens é uma etapa fundamental, onde são aplicadas técnicas como remoção de ruído, ajuste de contraste e redimensionamento. Essas metodologias são cruciais, pois preparam as imagens de maneira mais eficiente e precisa, garantindo que estejam otimizadas para uso nos algoritmos subsequentes.

1.0.1 Exemplos de bibliotecas utilizadas nessa etapa:

- **OpenCV:** Uma das bibliotecas mais utilizadas para processamento de imagens.
- **PIL/Pillow:** Biblioteca Python para manipulação de imagens.
- **Scikit-image:** Algoritmos para processamento de imagens.

1.0.2 Exemplos de aplicações:

- **Redimensionamento de imagem e conversão para cor cinza**

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

imagem = cv2.imread('/content/Monkey_D._Luffy_Anime_Post_Timeskip_Infobox.webp')

img_redimensionada = cv2.resize(imagem, (200, 300))

imagem_cinza = cv2.cvtColor(img_redimensionada, cv2.COLOR_BGR2GRAY)

imagem_cinza_rgb = cv2.cvtColor(imagem_cinza, cv2.COLOR_GRAY2BGR)

comparacao = np.hstack((img_redimensionada, imagem_cinza_rgb))

cv2_imshow(comparacao)

cv2.imwrite('comparacao_antes_depois.jpg', comparacao)
```

Esse código redimensiona uma imagem, converte para tons de cinza, cria uma comparação visual entre a imagem original e a imagem em tons de cinza, e salva essa comparação em um arquivo.

2 O que é segmentação de imagens?

Segmentação de imagens é o processo de subdividir uma imagem com base em segmentos específicos como cor, textura e intensidade. O objetivo é simplificar ou alterar a representação de uma imagem em algo mais significativo e fácil de analisar. É uma etapa fundamental em muitos sistemas de visão computacional e aplicações de reconhecimento de padrões. Tipos de segmentação incluem Segmentação Baseada em Limiarização, Região, Contorno, e Modelo.

2.0.1 Bibliotecas ideais para cada tipo de segmentação

- **OpenCV:** Para Segmentação Baseada em Limiarização.
- **Scikit-image:** Para Segmentação Baseada em Região.
- **OpenCV (cv2.Canny):** Para Segmentação Baseada em Contorno.
- **TensorFlow, Keras, PyTorch:** Para Segmentação Baseada em Modelos.

2.1 Exemplos de aplicações

- **Limiarização**

```
import cv2

_, imagem_segmentada = cv2.threshold(imagem_cinza, 127, 255, cv2.THRESH_BINARY)
```

Esse código aplica a técnica de limiarização para separar regiões de uma imagem com base em um valor de limiar. Pixels com valores maiores que 127 se tornam brancos (255), e pixels com valores menores ou iguais a 127 se tornam pretos (0).

- **Região**

```
import cv2
import numpy as np
from skimage import morphology, filters

limiar = filters.threshold_otsu(imagem_cinza)
imagem_segmentada = imagem_cinza > limiar

imagem_segmentada = morphology.remove_small_objects(imagem_segmentada, min_size=500)
imagem_segmentada = morphology.binary_closing(imagem_segmentada, morphology.disk(10))
```

Esse código segmenta a imagem usando um limiar automático e melhora a segmentação removendo objetos pequenos e preenchendo buracos.

- **Contorno**

```
import cv2

bordas = cv2.Canny(imagem_cinza, 100, 200)
```

Esse código detecta as bordas na imagem usando o algoritmo de Canny, com limiares de histerese de 100 e 200.

- **Modelo**

```
from tensorflow.keras.models import load_model
import numpy as np
import cv2
import matplotlib.pyplot as plt

modelo = load_model('/content/modelo_segmentacao.h5')

imagem_path = '/content/Monkey_D._Luffy_Anime_Post_Timeskip_Infobox.webp'
imagem = cv2.imread(imagem_path)
imagem = cv2.cvtColor(imagem, cv2.COLOR_BGR2RGB)
imagem = cv2.resize(imagem, (256, 256))
imagem = np.expand_dims(imagem, axis=0) / 255.0

segmentacao = modelo.predict(imagem)[0]
segmentacao = np.squeeze(segmentacao)

plt.imshow(segmentacao, cmap='gray')
plt.title('Segmentao␣com␣Modelo')
plt.axis('off')
plt.show()
```

Essa técnica utiliza modelos de aprendizado de máquina, como redes neurais convolucionais (CNNs), para identificar e classificar diferentes regiões ou objetos em uma imagem.

3 O que é detecção e classificação de imagens?

- **Detecção:** Essa etapa consiste em localizar um objeto específico em uma imagem, seja um rosto ou um objeto. A tarefa é complexa, pois não só identifica o objeto, como também sua posição através de caixas delimitadoras (bounding boxes).
- **Classificação:** Responsável por rotular a imagem completa, indicando a qual categoria ela pertence (por exemplo, "gato" como "animal").

3.0.1 Exemplos de bibliotecas/frameworks

- **TensorFlow:** Uma das bibliotecas mais populares para detecção e classificação de imagens.
- **PyTorch:** Um framework de deep learning muito usado, especialmente em pesquisa e desenvolvimento.
- **OpenCV:** Focada em aplicações de visão computacional, útil para pré-processamento de imagens e algumas tarefas básicas de detecção de objetos.
- **Keras:** API em Python que roda sobre TensorFlow, facilitando a construção e treinamento de modelos de deep learning para classificação de imagens.

3.1 Exemplos de aplicações

1. **Classificação de Imagens com Keras e TensorFlow**

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_labels))
```

Esse código utiliza o conjunto de dados CIFAR-10 e constrói uma Rede Neural Convolucional (CNN) para realizar a classificação de imagens.

2. **Detecção de Objetos com OpenCV e Haar Cascades**

```
import cv2

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

image = cv2.imread('path_to_image.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Neste exemplo, foi utilizado o Haar Cascade, um classificador pré-treinado para detectar rostos em imagens. Convertendo a imagem para escala de cinza e aplicando o detector, foi obtido as coordenadas dos rostos na imagem e desenhamos caixas ao redor dele

4 Outputs dos códigos

Exemplo de resultado usando as tecnicas levantadas na sessão 1 e 2 (Redimensionamento de imagem e conversão para cor cinza, e técnicas de Limiarização, Região e Contorno)

Figure 1: Redimensionamento de imagem e conversão para cor cinza

Figure 2: Aplicando as técnicas de Limiarização, Região e Contorno, respectivamente.