CURSO ATLÂNTICO-MACHINE LEARNING

ATIVIDADE 03- VISÃO COMPUTACIONAL

Nome: Antonio Pedro de Moura Laureno

Professor: Renê Ripardo Calixto

PRÉ-PROCESSAMENTO DE IMAGENS

1) Introdução

O pré-processamento de imagens é uma das etapas mais importantes para o processamento adequado de uma imagem e para a visão computacional. Tal fase tem o intuito de aprimorar a qualidade da imagem analisada para a boa implementação das técnicas seguintes, como a análise, o reconhecimento, a classificação etc. Essa etapa envolve uma série de fases em que a imagem analisada é submetida. A título de ilustração, as etapas citadas anteriormente são: Conversão para a escala de cinza; realce de contraste; redução de ruído; segmentação; eliminação de artefatos; correção de distorções; normalização e mapeamento de cores; redimensionamento e padronização.

2) Bibliotecas/Frameworks

Para a execução adequada da etapa de pré-processamento, existem muitas bibliotecas e frameworks que executam com competência tal fase. No entanto, no curso de machine learning do AtlantiBootcamp, foi utilizada a Linguagem Python e, para realizar o pré-processamento, foi utilizada a biblioteca **OpenCV**. Tal ferramenta oferece uma gama de funcionalidades para essa etapa da visão computacional, como a manipulação de imagens, a conversão de cores, a detecção de bordas etc. A título de exemplificação, também existem ferramentas na linguagem Java que trabalha com o pré-processamento de imagens, como a JAI (Java Advanced Imaging).

3) Aplicações

Para exemplificar melhor essa etapa, aqui está um código que exemplifica uma conversão para a escala de cinza utilizando o Python e a biblioteca OpenCV:

```
import cv2

# Essa etapa carrega uma imagem colorida

imagem = cv2.imread('imagem.jpg')

# Essa etapa converte a imagem para escala de cinza

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

# Por fim, essa etapa exibe a imagem em escala de cinza

cv2.imshow('Imagem em Escala de Cinza', imagem_cinza)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

SEGMENTAÇÃO DE IMAGENS

1) Introdução

A segmentação de imagens é um processo imprescindível para o devido processamento de imagens e visão computacional, que consiste em segmentar uma imagem em regiões ou objetos distintos com base em características visuais, como cores, intensidades de pixel, texturas ou bordas. Tal processo tem o intuito de facilitar a análise e a extração de informações ao separar partes específicas da imagem. A título de ilustração, uma das técnicas mais importantes da segmentação de imagens é segmentação por limiarização, onde os pixels com intensidade acima do limiar são atribuídos em uma região, enquanto os pixels de menor intensidade são atribuídos a outra região.

2) Bibliotecas/Frameworks

Para a execução adequada da etapa de segmentação de imagens, existem muitas bibliotecas e frameworks que executam com competência tal fase. Como citado anteriormente em pré-processamento de imagens, a biblioteca OpenCV também é usada para a segmentação de imagens. Além dessa ferramenta, é usada uma biblioteca em python é a SimpleCV, que é uma biblioteca de visão computacional com funcionalidades de segmentação.

3) Aplicações

Para exemplificar melhor essa etapa, aqui está um código que exemplifica uma segmentação por limiarização utilizando o Python e a biblioteca OpenCV:

```
import cv2

import numpy as np


# Carregar a imagem escolhida

image = cv2.imread('imagem.jpg', cv2.IMREAD_GRAYSCALE) # Carregar a imagem em escala de cinza obtida no pré-processamento

# Aplicar limiarização com a função “cv2.threshold” para segmentar a imagem
_, segmented_image = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)


# Exibir a imagem segmentada

cv2.imshow('Imagem Segmentada', segmented_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

DETECÇÃO/CLASSIFICAÇÃO DE IMAGENS

1) Introdução

A detecção e a classificação das imagens são as etapas finais e mais importantes na visão computacional e no Machine Learning. A detecção consiste na identificação e na localização de um objeto de interesse dentro de uma imagem. Já a classificação é a tarefa de atribuir uma classe ou um rotulo para determinada imagem como, por exemplo, detectar que a aquela imagem é uma “bola”, ou um livro etc. Para a execução precisa dessas duas etapas, é preciso desenvolver uma Rede Neural Convulacional (CNN), que consiste em um modelo de Deep Learning que é essencial para a visão computacional.

2) Bibliotecas/Frameworks

Para a execução adequada das etapas detecção e classificação, existem muitas bibliotecas e frameworks que executam com competência tais fases. a biblioteca TensorFlow do Python é amplamente usada para a detecção/classificação de imagens. Além dessa ferramenta, também existe uma biblioteca TensorFlow em JavaScript, a TensorFlowJs.

3) Aplicações

Para exemplificar melhor essa etapa, aqui está um código que exemplifica uma detecção/classificação utilizando o Python e a biblioteca TensorFlow:

```
import numpy as np

import tensorflow as tf

from PIL import Image

from object_detection.utils import ops as utils_ops

from object_detection.utils import label_map_util

from object_detection.utils import visualization_utils as vis_util

# Carregando o modelo pré-treinado e rótulos

MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'

PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

PATH_TO_LABELS = 'data/mscoco_label_map.pbtxt'

# Nessa etapa, é chamado o grafo do modelo

detection_graph = tf.Graph()

with detection_graph.as_default():

od_graph_def = tf.compat.v1.GraphDef()

with tf.compat.v2.io.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:

serialized_graph = fid.read()

od_graph_def.ParseFromString(serialized_graph)
```

```
    tf.import_graph_def(od_graph_def, name='')

    # Carregar o mapa de rótulos

    category_index
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)


# Crie uma Função para detecção e classificação de objetos

                def detect_and_classify_objects(image_path):
                    with detection_graph.as_default():
                        with tf.compat.v1.Session(graph=detection_graph) as sess:
                            image = Image.open(image_path)
                            image_np = np.array(image)
                            image_np_expanded = np.expand_dims(image_np, axis=0)

                            ops = detection_graph.get_operations()
                            all_tensor_names = {output.name for op in ops for output in op.outputs}
                            tensor_dict = {}
                            for key in ['num_detections', 'detection_boxes', 'detection_scores',
        'detection_classes']:
                                tensor_name = key + ':0'
                                if tensor_name in all_tensor_names:
                                    tensor_dict[key]                                    =
        detection_graph.get_tensor_by_name(tensor_name)

                            output_dict = sess.run(tensor_dict, feed_dict={'image_tensor:0':
        image_np_expanded})

                            vis_util.visualize_boxes_and_labels_on_image_array(
                                image_np,
                                output_dict['detection_boxes'][0],
                                output_dict['detection_classes'][0].astype(np.int32),
                                output_dict['detection_scores'][0],
                                category_index,
                                use_normalized_coordinates=True,
                                line_thickness=8)

                            Image.fromarray(image_np).show()

                # Passos para a imagem que se deseja processar
                image_path = 'path_to_your_image.jpg'  # Substitui pelo caminho da imagem
                detect_and_classify_objects(image_path)
```