LabIC Images Segmentation

Created on Fri Apr 14 19:04:14 2023

@author: Labic

class **labic_images_segmentation.Dataset**(*folder: str, norm_imgs_folder: str, gt_folder: str, ORIGINAL_SIZE=None, NEW_SIZE=None*)

A classe Dataset contém todos os processos vinculados ao carregamento e separação dos dados. Após a inicialização, teremos:

- No atributo X: todas as imagens;
- No atributo Y: todas as máscaras;

load_images()

Organiza a lista das imagens no diretório e chama a função load_images_array para alimentar os atributos X, Y e img_shape da classe.

load_images_array(*img_list: list, original_size=160, new_size=None*)

Recebe um glob das imagens e converte em um numpy array no formato que o Keras aceita.

Parâmetros:
- **img_list** (*list*) – Lista com todos os nomes das imagens no diretório.
- **new_size** (*int*) – Novo size da imagem (largura e altura).

Retorno: Conjunto de imagens no formato de input do Keras [(exemplo formato Keras: (5, 256, 256, 1)]

e uma tupla com a altura e a largura, respectivamente, da imagem original. :rtype: tuple

resize_one_img(*img: ndarray, width: int, height: int*)

Redimensiona uma imagem.

Parâmetros:
- **img** (*numpy.ndarray*) – Imagem original.
- **width** (*int*) – Nova largura da imagem.
- **height** – Nova altura da imagem.

Retorno: Imagem redimensionada.

Tipo de retorno: numpy.ndarray

split_dataset(*seed_min=0, seed_max=1048576, test_size=0.2*)

Separa as imagens e as máscaras de treino e validação. Alimenta os atributos X_train, Y_train, X_val e Y_val.

Parâmetros:
- **seed_min** (*int*) – Valor mínimo para a semente do random.
- **seed_max** (*int*) – Novo size da imagem (equivalendo para largura e altura).
- **test_size** (*float*) – Tamanho do conjunto de teste [Valores entre 0 e 1. Ex.: 0.2 = 20% do total dos dados para teste].

---

class **labic_images_segmentation.DataAugmentation**(*X_train: ndarray, Y_train: ndarray, use_batch_size: int, X_val: ndarray, Y_val: ndarray, factor=0.2, direction='horizontal', rotation=0.1*)

A classe contém todas as funções necessárias para o processo de Data Augmentation.

- No atributo trainDS: dados de treino após o processo de data augmentation;
- No atributo valDS: dados de validação após o processo de data augmentation;

augmentation()

Aplica o processo de data augmentation de acordo com os parâmetros repassados.

Retorno: Retorna os dados de treino e validação após o Data Augmentation.

Tipo de retorno:

---

class **labic_images_segmentation.SegmentationModel**(*N: int, segmentation_model: str, backbone_name: str, trainDS, valDS, epochs: int, callback=None, input_layer_shape=None*)

Após a inicialização, teremos:

- No atributo model: Modelo da rede;
- No atributo history: Report do treinamento;

generate_model()

Gera o modelo. Alimenta os atributos model e history.

Retorno: Modelo e History

Tipo de retorno: keras.engine.functional.Functional, keras.callbacks.History

---

class **labic_images_segmentation.SaveReport**(*model, history, folder_name: str, n_fold: int, epochs: int, exec_folder_name: str, use_batch_size=4*)

Salva o modelo e o history (report).

create_folder(*dirName*)

Cria o diretório se não existir.

Parâmetros: **dirName** (*str*) – Nome do diretório a ser criado.

organize_folders()

Organiza os diretórios. Dentro do output_folder haverá (ou será criada) a pasta outputs. Dentro da pasta outputs, por sua vez, estarão os diretórios do conjunto de execuções (por padrão, iniciados com Exec_). Dentro de cada pasta de execução, estarão os diretórios de cada n execução. [Ex. Path: output_folder/outputs/Exec_folder/fold_n]

Retorno: Momento da execução, path da pasta da execução de número n, path da pasta do conjunto de execuções.

Tipo de retorno: str, str, str

save_history()

Salva o history do treinamento em formato .csv e .txt O arquivo .txt necessário pois, apesar do formato .csv facilitar o carregamento de dados a posteriori, o armazenamento neste pode ter falhas de caracteres, sobretudo em valores muito altos na Loss.

save_model()

Alimenta o atributo n_fold_folder_name, organiza as pastas chamando a função organize_folders. Salva o modelo no arquivo .h5.

---

class **labic_images_segmentation.PredictImages**(*test_images, n_fold_folder_name: str, model_name: str, use_batch_size: int, img_shape: tuple*)

Carrega os dados de teste e realiza a predição.

predict()

Carrega o modelo, cria o diretório outputs_prod dentro da pasta da execução n. Realiza e salva as predições, redimensionando as images no momento de salvar.

---

class **labic_images_segmentation.DiceCoef**(*gt_imgs: ndarray, pred_folder: str, new_size: int*)

Após a inicialização, teremos:

- No atributo pred_imgs: Imagens de teste (em formato np.ndarray);
- No atributo img_shape: Tupla com dimensões das imagens;
- No atributo dice: Valor do dice entre máscaras de teste e predições.

dice_coef(*y_true: ndarray*, *y_pred: ndarray*)

Calcula o dice entre as máscaras de teste e as predições.

Parâmetros:
- **y_true** (*numpy.ndarray*) – Máscaras das imagens de teste.
- **y_pred** (*numpy.ndarray*) – Predições do modelo.

Retorno: Coeficiente Dice

Tipo de retorno: float

generate_csv_dice(*n_all_folders: int*, *save_report*, *title: str*)

Cria um arquivo .csv no diretório do conjunto de execuções com o valor de todos os dices dos n_folds. Alimenta o atributo df com os mesmos dados do csv.

Parâmetros:
- **n_all_folders** (*int*) – Número total de execuções no conjunto.
- **save_report** (*labic_images_segmentation.SaveReport*) – Objeto da classe SaveReport para organização dos arquivos e diretórios.
- **title** (*str*) – Título do arquivo .csv. Recomenda-se indicar modelo, backbone, entre outras informações relevantes.

generate_graphic(*epochs: int*, *segment*, *save_report*, *graphic_type: str*)

Gera os gráficos com dados do History. Armazena as imagens em formato png.

Parâmetros:
- **epochs** (*int*) – Número de épocas.
- **segment** (*labic_images_segmentation.SegmentationModel*) – Objeto da classe SegmentationModel.
- **save_report** (*labic_images_segmentation.SaveReport*) – Objeto da classe SaveReport.
- **graphic_type** (*str*) – Tipo do gráfico («iou_score» ou «loss»).

save_dice(*adress: str*)

Salva o valor do dice na pasta da execução n.

Parâmetros: **adress** (*str*) – Path para arquivo .txt do dice.