Documentação Projeto Final de Programação

Artur Albuquerque Silva
Número de Matricula 1921193

Dezembro 2020

1 Especificação do programa

O programa descrito neste documento tem o objetivo de realizar a segmentação semântica de imagens de tomografia computadorizada de pulmões para detectar infecções pulmonares. O programa funciona passando um arquivo de configuração, ele então treina, testa e salva os resultados da segmentação semântica. Segmentação semântica consiste no processo de marcar cada pixel de uma imagem com uma classe correspondente como visto na Figure 1.

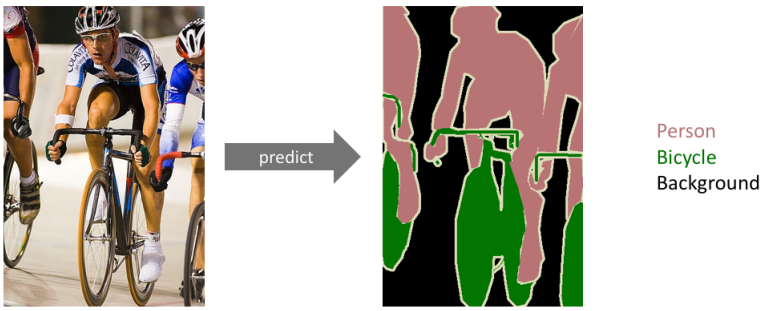

Figure 1: Exemplo de segmentação semântica.
Fonte: `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#devkit`

O programa foi escrito em Python e se encontra no GitHub em: `https://github.com/astobiro/semantic_segmentation_project`. Na pagina do GitHub há uma base já preparada para uso, mas as instruções para preparar a base se encontram na seção 4. A base preparada foi obtida no kaggle em: `https://www.kaggle.com/andrewmvd/covid19-ct-scans`. Ela consiste em 20 exames de tomografia computadorizada de pacientes confirmados com COVID-19 e as marcações de especialistas tanto dos pulmões quanto das infecções.

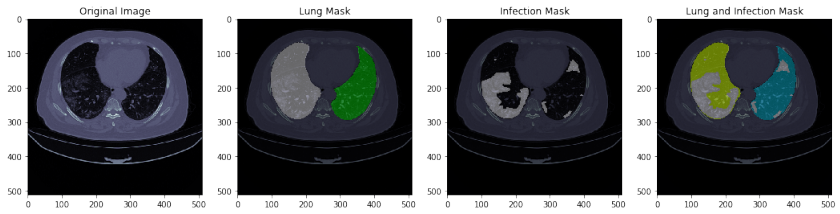

Figure 2: Exemplo da base de dados. Na ordem a imagem original, a imagem com somente o pulmão marcado, a imagem somente com a infecção marcada e a imagem com o pulmão e infecção marcados

2 Projeto do programa

Nesta seção serão descritos as classes utilizadas no programa e também o diagrama de execução.

2.1 data_loader

Classe que carrega o dataset em batches para melhor utilizar a memoria.

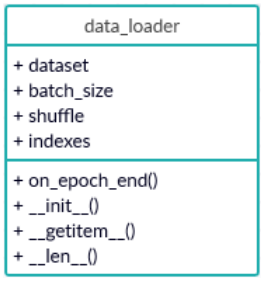

Figure 3: Diagrama de classe de data_loader

Possui as seguintes variáveis:

- dataset : recebe um objeto do tipo dataset (descrito abaixo)
- batch_size : inteiro que recebe o tamanho de imagens no batch
- shuffle : booleano que se 'True' embaralha os índices a cada época
- indexes : np array que recebe o tamanho do dataset

Possui as seguintes funções:

- on_epoch_end() : Função de callback para embaralhar os indexes a cada época.

- __init__() : Função que é responsavel pela inicialização do objeto, é chamada quando chamamos da forma $X = Dataloder(dataset,\ batch\_size)$.
- __getitem__() : Função que é responsável pelo selecionamento de um item dentro do objeto (é chamada quando por exemplo queremos o item i do data_loader com variável A e colocamos no codigo $A[i]$).
- __len__() : Função que é responsável por retornar o numero de batches por época do objeto (Tem o funcionamento parecido com o de __getitem__() só que funciona quando colocamos $len(A)$).

2.2 dataset

Classe que carrega o dataset a partir de duas pastas e um dataframe.

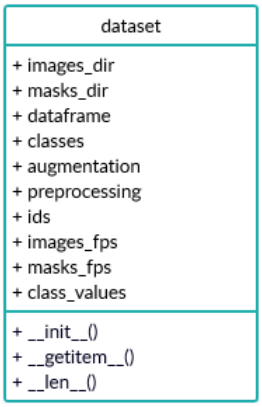

Figure 4: Diagrama de classe de dataset

Possui as seguintes variáveis:

- images_dir : Guarda o caminho para a pasta com as imagens.
- masks_dir : Guarda o caminho para a pasta com as mascaras.
- dataframe : Guarda um objeto do tipo dataframe que guarda o nome de cada imagem de determinado conjunto (treino, teste ou validação), tendo um dataframe para cada.
- classes : Guarda as classes em forma de lista.
- augmentation : Variável opcional, guarda a função utilizada para realizar o aumento da base.
- preprocessing : Variável opcional, guarda a função utilizada para realizar preprocessamento.

- ids : Guarda os nomes das imagens extraído do dataframe
- images_fps : Guarda as imagens em forma de lista, já carregadas usando opencv.
- masks_fps : Guarda as máscaras em forma de lista, já carregadas usando opencv
- class_values : Guarda os valores de classe em lista, já atribuídos para cada imagem.

Possui as seguintes funções:

- __init__() : Função que é responsavel pela inicialização do objeto, é chamada quando chamamos da forma $X = Dataset(images\_dir,\ masks\_dir,\ datafram,\ classes)$.
- __getitem__() : mesmo proposito da __getitem__() de data_loader.
- __len__() : Retorna o tamanho da lista de imagens quando utilizado da forma $len(A)$.

2.3 dataset_prep

Classe que prepara o dataset e carrega o arquivo de configurações e dataframes respectivos para treino, teste e validação.

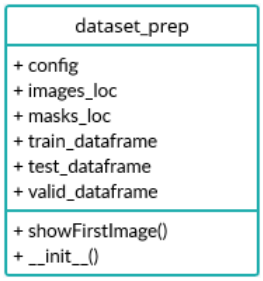

Figure 5: Diagrama de classe de dataset_prep

Possui as seguintes variáveis:

- config : Guarda uma variável carregada de um arquivo json, é o arquivo com as configurações da rede.
- images_loc : Guarda uma string com o caminho para a pasta onde estão as imagens.
- masks_loc : Guarda uma string com o caminho para a pasta onde estão as mascaras.

- train_dataframe : Guarda o dataframe com o nome das imagens de treino carregado a partir de um csv.
- test_dataframe : Guarda o dataframe com o nome das imagens de teste carregado a partir de um csv.
- valid_dataframe : Guarda o dataframe com o nome das imagens de validação carregado a partir de um csv.

Possui as seguintes funções:

- showFirstImage() : Função para mostrar a primeira imagem de um determinado conjunto, recebe como argumento 0, 1 ou 2 que indica para mostra a imagem de treino, validação e teste respectivamente.
- __init__() : Função que é responsável pela inicialização do objeto, é chamada quando chamamos da forma $X = Dataprep(config)$.

2.4 segmentation_model

Classe que é responsável por carregar o modelo, treinar e mostrar os resultados.

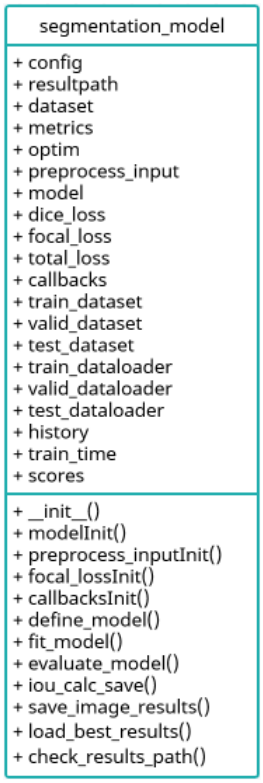

Figure 6: Diagrama de classe de segmentation_model

Possui as seguintes variáveis:

- config : Guarda uma lista com as configurações da rede.
- resultpath : Guarda em uma string o caminho para os arquivos de resultado a partir da função check_results_path().
- dataset : Guarda um objeto do tipo Dataset descrito acima.
- metrics : Guarda uma lista com as métricas que serão utilizadas (no caso IOU e F-score).
- optim : Guarda um otimizador utilizado (no caso Adam).
- preprocess_input : Guarda o tipo de preprocessamento que será usado pelo objeto dataset a partir da função preprocess_inputInit().
- model : Guarda um objeto do tipo model da biblioteca keras a partir da função modelInit().
- dice_loss : Guarda um objeto do tipo losses.DiceLoss da biblioteca segmentation_models.
- focal_loss : Guarda um objeto do tipo losses.BinaryFocalLoss ou losses.CategoricalFocalLoss a partir da função focal_lossInit().
- total_loss : Guarda a soma do dice_loss com focal_loss.
- callbacks : Guarda um objeto do tipo callbacks da biblioteca keras a partir da função callbacksInit().
- train_dataset : Guarda um objeto do tipo dataset que recebe como entrada o caminho das imagens e o dataframe com as imagens de treino.
- valid_dataset : Guarda um objeto do tipo dataset que recebe como entrada o caminho das imagens e o dataframe com as imagens de validação.
- test_dataset : Guarda um objeto do tipo dataset que recebe como entrada o caminho das imagens e o dataframe com as imagens de teste.
- train_dataloader : Guarda um objeto do tipo dataloader que recebe como entrada o objeto train_dataset e o batch_size.
- valid_dataloader : Guarda um objeto do tipo dataloader que recebe como entrada o objeto valid_dataset.
- valid_dataloader : Guarda um objeto do tipo dataloader que recebe como entrada o objeto test_dataset.
- history : Guarda um objeto do tipo history da biblioteca keras, resultado de rodar o fit_generator no modelo com entrada o train_dataloader, o número de epocas, valid_dataloader.

- train_time : Guarda o tempo de duração do treinamento após rodar a função fit_model().
- scores : Guarda o resultado de rodar o evaluate_generator no model com entrada o test_dataloader.

Possui as seguintes funções:

- __init__() : Função que é responsável pela inicialização do objeto, é chamada quando chamamos da forma *X = SegmentationModel(config, dataset)*.
- modelInit() : Função que é responsável por atribuir a variável model o objeto a partir da função Unet da biblioteca segmentation_models.
- preprocess_inputInit() : Função que é responsável por atribuir a variável preprocess_input o objeto a partir da função get_preprocessing da biblioteca segmentation_models.
- focal_lossInit() : Função que é responsável por atribuir a variável focal_loss o objeto BinaryFocalLoss se só há uma label possível (se é binario a segmentação) ou CategoricalFocalLoss se há mais que uma (se tem varias classes para segmentar).
- callbacksInit() : Função que é responsável por atribuir a variável callbacks o objeto de callback carregado.
- resultpath : Guarda em uma string o caminho para os arquivos de resultado a partir da função check_results_path().
- fit_model() : Função que é reponsável por realizar o treinamento do modelo, atribui as variáveis history e train_time e salva os gráficos do treinamento.
- evaluate_model() : Função que é responsável por atribuir a variável scores os valores retornados da funçao evaluate_generator em cima do modelo com o test_dataloader.
- iou_calc_save() : Função que é responsável por computar o IOU para cada imagem especifica e salvar em um csv.
- save_image_results() : Função que é responsável por salvar o resultado da rede em cima do conjunto de testes em imagens.
- load_best_results() : Função que é responsável por carregar os melhores pesos do treinamento no modelo.
- check_results_path() : Função que é responsável por inicializar a variável resultspath, checa se existe a pasta e a cria caso contrario.

2.5 main

Objeto responsável por chamar para execução todos os outros.

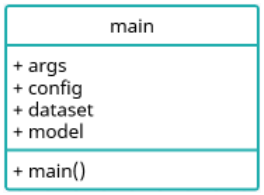

Figure 7: Diagrama de classe de main

Diagrama de execução em Figure 8.

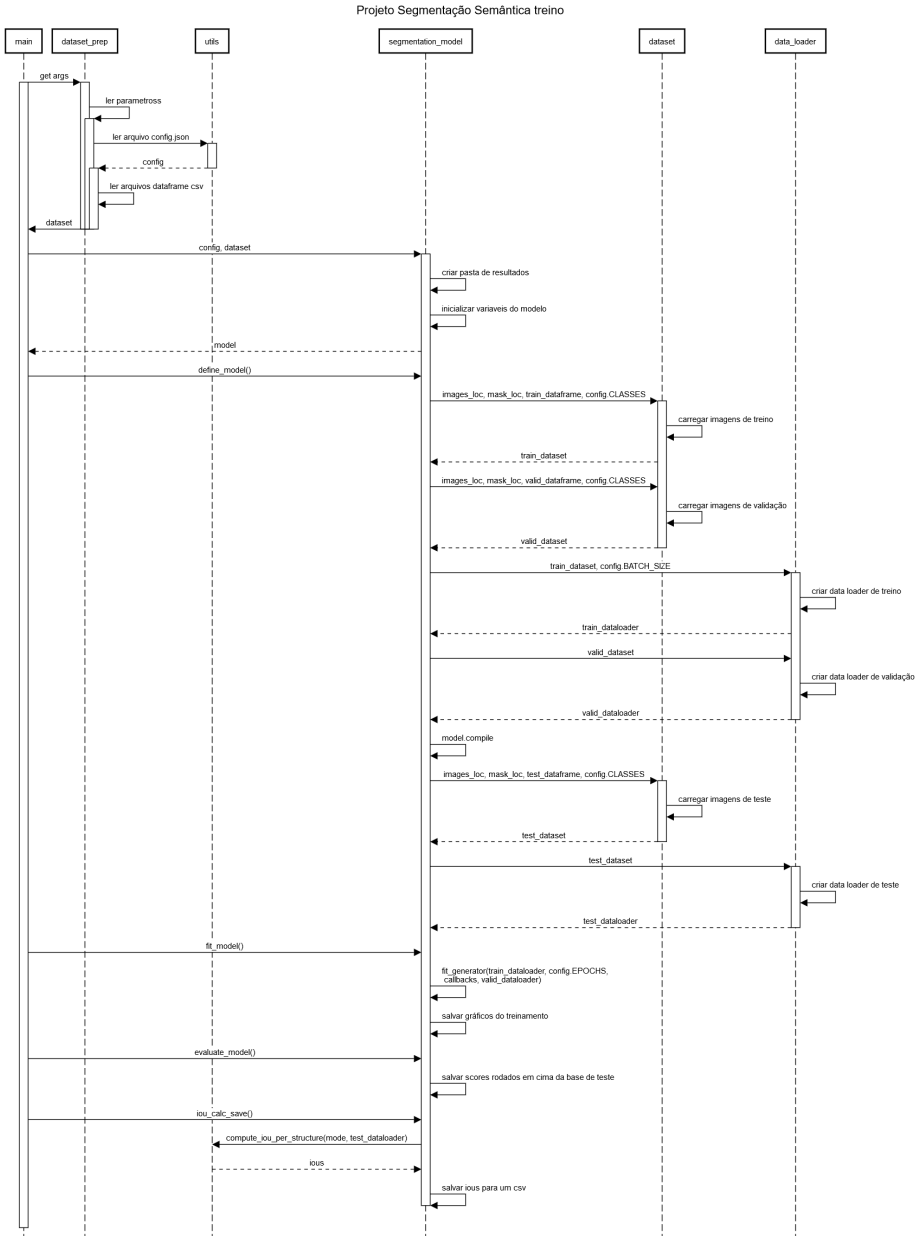

Figure 8: Diagrama de execução para treino

2.6 utils

O objeto utils guarda algumas funções utilizadas pelos outros objetos. Funções:

- visualize() : Função para mostrar uma imagem utilizando a biblioteca

matplotlib.

- plot_stuff() : Tem como entrada uma imagem e uma mascara, mostra a imagem e mascara sobre a imagem lado a lado.
- Params() : Função que lê o arquivo json de configuração e o coloca em um objeto utilizável.

2.7 iou_calc

O objeto iou_calc guarda as funçõs utilizadas para o calculo do IOU em uma imagem ou em várias. Suas funções são:

- round_if_needed() : se o valor for maior que o threshold passado colocar o valor para o threshold.
- compute_iou_per_class() : Calcula o IOU para cada classe dado uma ground truth e uma mascara.
- compute_iou_per_structure() : Dado um modelo e um test_generator chama a função compute_iou_per_class() para cada imagem predita do teste e guarda em uma lista.

3 Testes e resultados

Foi realizado um teste no programa para observar se estava tudo funcionando, os 20 pacientes foram separados em treino, validação e teste. As imagens foram convertidas da tomografia 3D para 2D nomeadas a partir do paciente e do numero da fatia. No total obtemos 3520 imagens em 384x384. Foram então criados 3 arquivos csv que continham o nome das imagens de treino, validação e teste.

Dessa maneira outros testes poderiam ser realizados sem haver a necessidade de modificar as imagens. O teste foi feito como descrito na Figure 8, os parâmetros utilizados se encontram no arquivo config.json. O treinamento durou aproximadamente 3 horas parou em 27 épocas e obteve um Loss de 0.000115752242, um IOU médio de 0.997600019 e um f1 score médio de 0.9987947345. A curva de aprendizado pode ser vista na Figure 9 e Figure 10

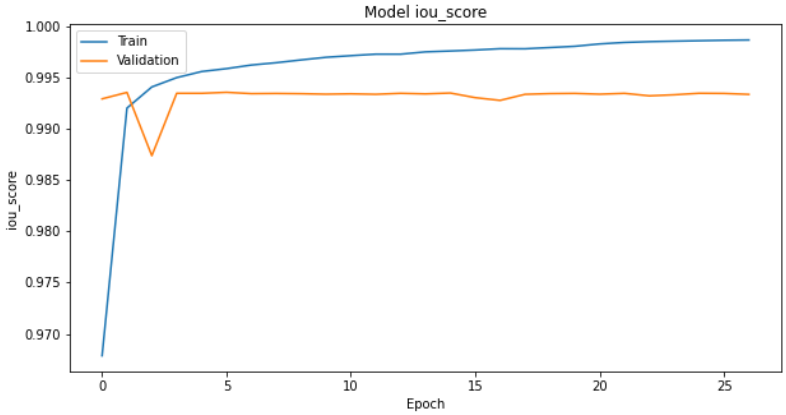

Figure 9: Gráfico de treinamento IOU

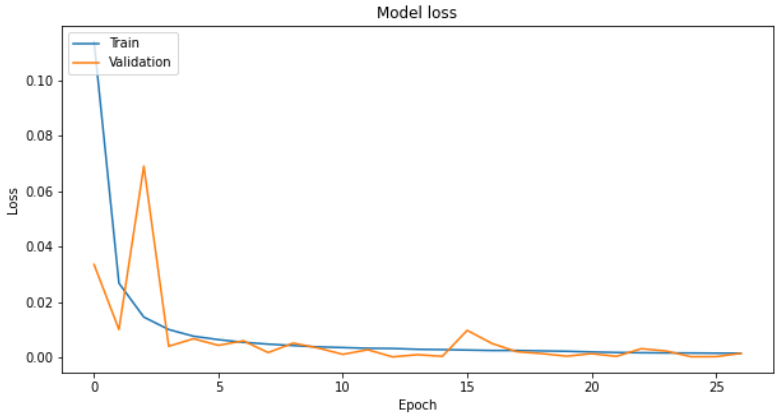

Figure 10: Gráfico de treinamento Loss

Algumas imagens das mascaras geradas:

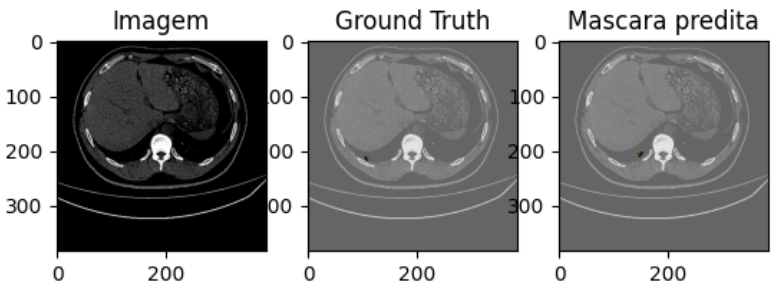

Figure 11: Exemplo 1

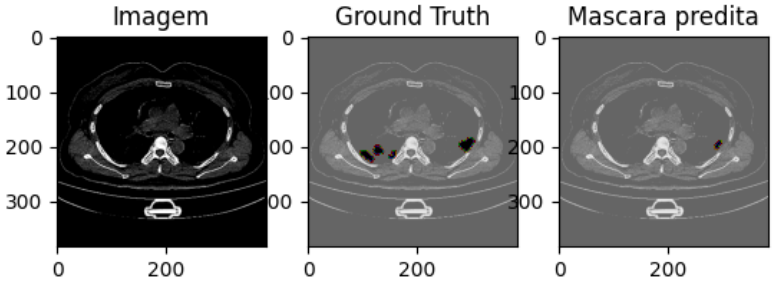

Figure 12: Exemplo 2

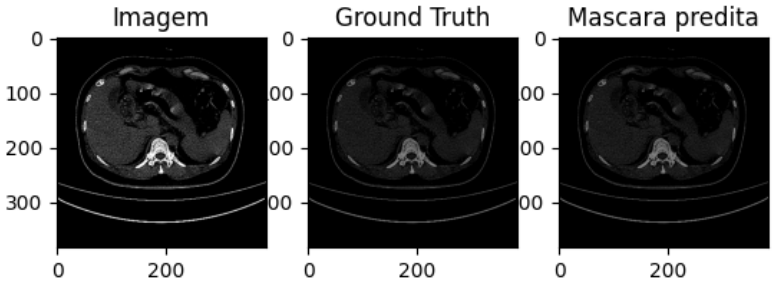

Figure 13: Exemplo 3

A maioria dos resultados ficaram como o Exemplo 3 (Figure 13) por isso o valor alto de IOU e F1 médio. O que poderia melhorar o resultado seria fazer um aumento da base. Outra coisa seria selecionar as fatias melhor para que a proporção de imagens com marcação e imagens sem marcação fique mais perto de 1:1.

4 Documentação para o usuário

Para utilizar o programa a primeira coisa que deve ser feita é instalar as dependências. As bibliotecas utilizadas são Segmentation Models, Albumentations, Keras (versão 2.3.1), TensorFlow (versão 2.1.0), Keras Applications (versão 1.0.8), Image Classifiers (versão 1.0.0) e Efficientnet (versão 1.0.0). Para instalar todas as bibliotecas pode usar os comandos:

```
pip install -U --pre segmentation-models
pip install keras==2.3.1
pip install tensorflow==2.1.0
pip install keras_applications==1.0.8
pip install image-classifiers==1.0.0
pip install efficientnet==1.0.0
```

Após pegar fazer um clone do repositorio do GitHub é só rodar por linha de comando a partir da pasta raiz do projeto (a pasta semantic_segmentation_project) o comando: **python src/main.py config.json**. O código ira executar e treinar os parametros do arquivo config.json. O arquivo config.json é este:

```
{
    "BACKBONE" : "efficientnetb3",
    "BATCH_SIZE" : 4,
    "CLASSES" : ["infection"],
    "LR" : 0.0001,
    "EPOCHS" : 40,
    "ACTIVATION" : "sigmoid",
    "imgfolder": "data/datasets/dataset2D-384x384/lungs",
    "maskfolder" : "data/datasets/dataset2D-384x384/masks",
    "train_dataframepath" :"data/datasets/dataset2D-384x384/n1_train_dataframe.csv",
    "test_dataframepath" :"data/datasets/dataset2D-384x384/n1_test_dataframe.csv",
    "valid_dataframepath" :"data/datasets/dataset2D-384x384/n1_valid_dataframe.csv",
    "TESTNO" : "n1",
    "IMAGE_SIZE" : 384,
    "load" : "n"
}
```

Para criar um arquivo de config customizado é só modificar este arquivo ou criar um arquivo tendo estes parâmetros todos. Os parâmetros são autoexplicativos exceto o parâmetro load, este parâmetro tem 2 estados n e y, quando é n o programa irá realizar o treinamento do 0. Quando é y o programa irá ler o arquivo de peso best_weights.h5 encontrado dentro da pasta data/results/TESTNO (TESTNO sendo o parâmetro dentro do arquivo). Esse arquivo é o arquivo de pesos do do ultimo treinamento. O programa irá então carregar os pesos e aplicar no modelo.

Para utilizar um dataset diferente o que é preciso é coloca-lo da forma imagens sem marcação em uma pasta chamada lungs, e as suas respectivas mascaras em outra pasta chamada masks. Os arquivos correspondentes de imagem e mascara devem ter o mesmo nome.

Para criar o dataframe de treino, teste e validação os arquivo csv tem a primeira coluna com o primeiro campo vazio e depois uma numeração indo de 0 a o numero máximo de imagens. A segunda coluna com o primeiro campo sendo filename e o resto sendo o nome das imagens que serão utilizadas. O mesmo deve ser feito para as imagens de teste e de validação. O arquivo csv deve ser nomeado TESTNO_train_dataframe.csv para o arquivo de treinamento

e trocando o train para test e valid para o arquivo de teste e validação respectivamente. O parâmetro IMAGE_SIZE deve ter o tamanho da dimensão das imagens tendo de modificar o arquivo segmentation_model se for utilizar uma imagem que não tem a mesma largura e altura. O parâmetro classes também terá de ser modificado para o caso especifico.