System Requirements Specification Index

For

Medical Image Segmentation for Tumor Detection

Problem Statement : Medical Image Segmentation for Tumor Detection

Description : Detecting polyp tumors using segmentation and deep learning involves using advanced computer algorithms to analyze medical images. Imagine your body is scanned, and the computer needs to find abnormal growths, like polyps. Deep learning models, inspired by the human brain, learn to recognize patterns in these images. Segmentation is like drawing boundaries around specific areas, isolating the polyps. This process helps doctors pinpoint and analyze potential issues more accurately. By leveraging these smart algorithms, medical professionals can swiftly and precisely identify polyps in images, aiding in early detection and improving the chances of successful treatment. It's like having a highly trained assistant for quicker and more accurate medical diagnoses.

The solution contains the following folder structure:

Polyp Segmentation |

|-- arguments.py

|-- dataloader.py

|-- loss.py

|-- main.py

|-- metric.py

|-- model.py

|-- test_functional.py

|-- TestCaseResultDto.py

|-- testing.py

|-- TestResults.py

|-- TestUtils.py

|-- train.py

|-- train_helper.py

|-- utils.py

|-- requirements.txt

|-- Commands-Readme.txt

The arguments.py is the main helper file that contains a single function to read command line arguments from terminal at the time of main file calling. The arguments include data

directory path, image folder name, mask folder name, batch size, number of epochs and a name for the experiment.

The dataloader.py file contains code for loading and augmenting data. It has the following class:

1. **Class PolypDatasetLoader(Dataset)** -> It is used to load and perform operation on the dataset. It inherits a features of “Dataset” class from torch.utils.data. It takes in 3 arguments: image directory path, mask directory path and size of the input image.

    a. **Function __augmentation** -> It is used to perform augmentation operation on the image like rotation and flipping to increase the size of the dataset used for training.

    b. **Function __make_dataset** -> It is used to create a list of addresses that are stored in memory for both the image and mask.

c. **Function __normalize_image** -> It is used to normalize the image that will help the model the model to understand and locate the polyp easily.

d. **Function __getitem__** -> It is a built-in function in the “Dataset” class that will read the image, resize it to the input size, perform the augmentation operation on it, and return the image. It does same for the mask.

e. **Function __len__** -> It is also a built-in function in the “Dataset” class that will return the count of samples in the dataset.

The loss.py file contains the code of the losses that are used to train the model. It has the following class:

1. **Class DiceBCELoss(nn.Module)** -> It is used to calculate the dice binary cross entropy loss on the predicted mask and the original mask. It first straightens both the masks and the calculate the intersection to get the dice loss. Then it performs the binary cross entropy (BCE) on the predicted mask and the original mask to get bce loss. It returns the sum of both the losses.

The main.py is the main runner file that is the initial point for the training the model. It has no classes, but it contains the code for following operations:

1. **Input Command Line Arguments**
2. **Dataset Splitting**
3. **Loading Data via data loader**
4. **Model and Hyperparameter Loading**
5. **Training the Model**
6. **CSV Record Keeping**

The metric.py has no classes, it only contains a function for calculating the dice coefficient in which the predicted and original masks were first flatten then the intersection was calculated. That intersection is then used to calculate the dice score which is then returned.

The model.py file contains the code for the architecture of the model. It has the following class:

1. **Class SELayer:** It stands for Squeeze and Excitation Block which first squeezes the input channel and then spares the input channels.

2. **Class ResidualBlock:** It uses skip connections to pass the information to forward layers.

3. **Class StridedConvBlock:** It uses a stride value as 2 to reduce the size of the image to half and also use batch normalization.

4. **Class EncoderBlock:** It is used for encoding the input image to get the understanding of the portion that contains tumor.

5. **Class CompNet:** It is the complete model which contains the encoder and a decoder. It takes the image of size 512x512 as input and results a binary mask as output.

The test_function.py file contains the code for testing the trained model on certain test cases and sending the results to the Yaksha server. It contains three functions that represents the three test cases for testing:

1. **Function test_model_exists:** Check whether the model exists on the specified path or not.

2. **Function original_predicted_mask_same_size:** Checks whether the predicted output and the original mask are of same size or not.

3. **Function metric_value_test:** Checks whether the Jaccard Coefficient and Dice Score for the model are above 80% or not.

The TestCaseResultDto.py file contains the code to take in arguments and convert those argument to dictionary.

The test.py file contains the code for testing the pretrained model for new images. It has the following class:

1. **Class Test ->** This class is built for testing the model output on the dataset set and saving that output in a certain specified folder. It has following functions:

a. **Function load_model:** It is used to load the model and its weights from the saved model file.

b. **Function load_dataset:** It is used to load the dataset in the memory.

c. **Function __metric_calculation:** It is used to calculate the jaccard coefficient and dice score metric between predicted and original mask.

d. **Function __denormalize_image:** It is used to denormalize the image to make it look similar to the original image.

e. **Function __save_image_func:** It is used to convert the tensor image to numpy image and save it.

f. **Function __save_img:** It is used to divide the batch images to images and then pass that image to save function.

g. **Function test:** It is used to run the model on the test image dataset and save the loss, jaccard coefficient and dice score values.

2. **Function test_helper:** This is a helper function that will make the object of the **class Test** and call the specific methods from that class in order for testing the model.

The TestResults.py file contains the code to take in arguments and convert those argument to dictionary.

The TestUtils.py file contains the code that connects to the Yaksha server and send the test case results to the server.

The train_helper.py file helps the main file in running the model for training by initializing hyperparameters, running the model for the specific epochs, scheduling the learning rate on the validation losses and saving the model with best jaccard coefficient value. This class is given below:

Class TrainHelper: This class take the model and the device type as arguments. It has following functions:

1. **Function __initialize_hyperparameters:** This function is used to initialize hyperparameters like optimizers, schedulers and the loss function.

2. **Function run:** This function takes the command line arguments, train dataloader, validation dataloader and destination directory as input. It's purpose is to run the model for given number of epochs, schedule the learning rate on the validation losses, saving the model with best jaccard coefficient value and saving all the values into a CSV file.

The train.py file contains the code for training the model on the given set of images. It has the following class:

1. **Class TrainingModule ->** This class is built for supporting the training and evaluation of the model on the respective datasets. It has following functions:

    a. **Function __metric_calculation:** It is used to calculate the jaccard coefficient and dice score metric between predicted and original mask.

    b. **Function train:** It is used to train to the model on the image dataset, calculate the bce dice loss and backpropagate the loss to update the weights which will help the model in locating the tumor. It also calls for metric calculation and returns all the values.

c. **Function evaluation:** It is used to evaluate to the trained model on the image dataset, calculate the bce dice loss. It also calls for metric calculation and returns all the values.

d. **Function save:** It saves the model at the specified path.

The utils.py file contains the helper functions for creating destination folder, splitting the dataset into groups and loading data. The following functions are present in the file:

1. **Function folder_creation:** It take the name of the destination folder as input argument and creates the folder with a unique name with datetime attached to it.

2. **Function splitting_data:** It takes the command line arguments as input for splitting the data into training and validation set and return the save directory location.

3. **Function get_dataloader:** It takes the command line arguments and the new split data directory location as input and pass it to dataset loader to read the image and transform it as well as read the mask on the call of the dataloader variable.

Important files for training and testing:

1. For Training -> main.py
2. For Testing -> test.py

Execution Steps to Follow:

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. To setup environment:

   pip install -r requirements.txt
5. To train the model type:

   python3 main.py -d “<<your dataset path>>” -i “<<image folder name>>” -m “<<mask folder name>>” -b <<batch size for training>> -e <<number of epochs to train>> -n “<<name of the result folder>>”

   Example: python3 main.py -d “data/PNG” -i “Original” -m “Ground Truth” -b 2 -e 100 -n “polyp_segmentation_results”
6. To run Test cases:

   python3 testing.py

   Example: python3 testing.py